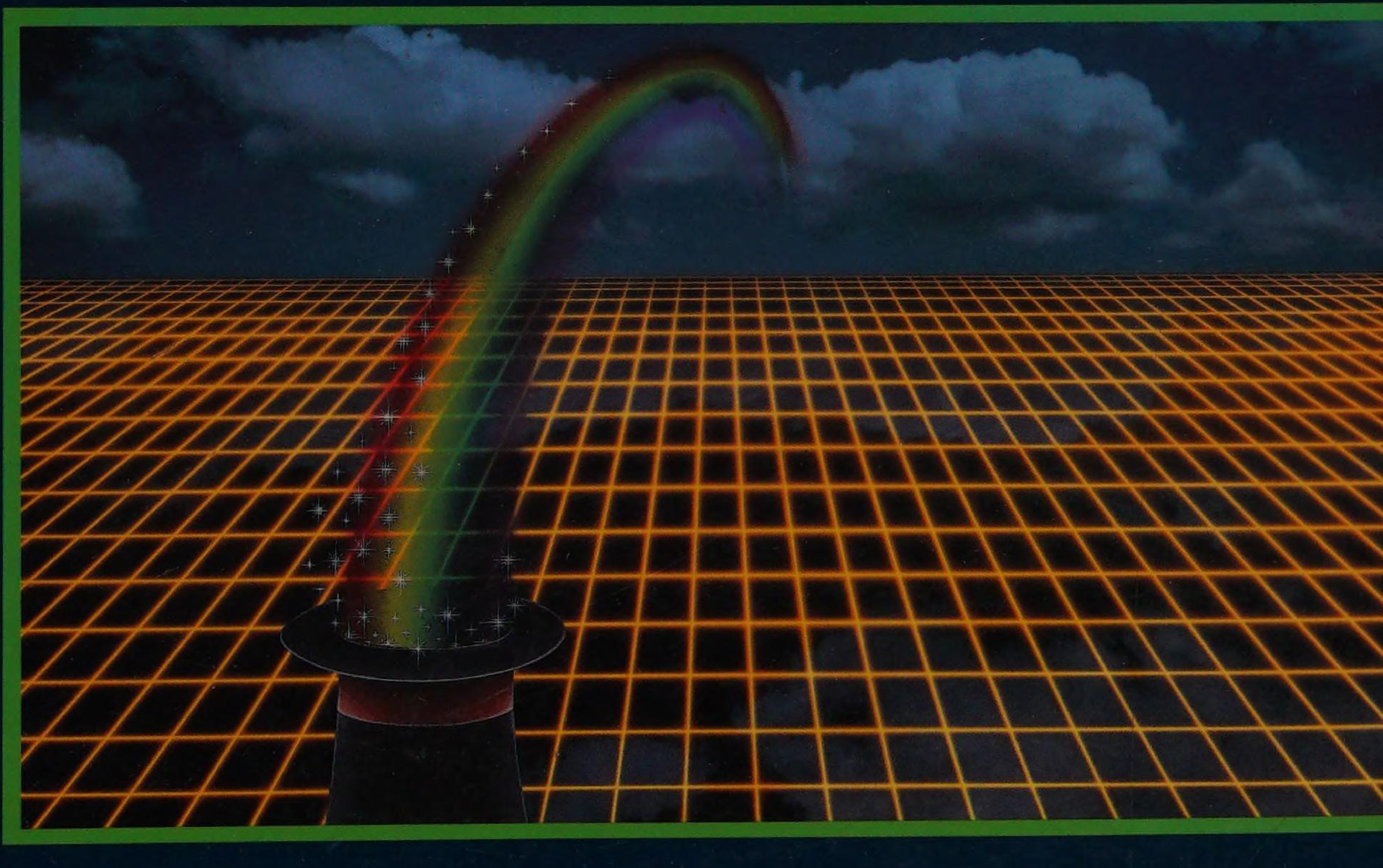
WILEY

$ 3.00

DOS PRODUCTIVITY TIPS AND TRICKS



GILBERT HELD

DOS Productivity Tips & Tricks

Related Titles of Interest from John Wiley & Sons

OS/2: A Business Perspective, Conklin
OS/2: Features, Functions, and Applications, Krantz, Mizell, Williams
The New Dos 4.0, Christopher, Feigenbaum, and Saliga
PC Upgrader's Manual: How to Build and Extend Your System, Held
IBM AS/400: A Business Perspective, Hoskins
Hard Disk Smarts: Everything You Need to Know to Choose and Use Your Hard Disk, Bosshardt
Ventura Publisher for the IBM PC: Mastering Desktop Publishing, Jantz
Advanced WordPerfect: A Power User's Guide, Farrell
R:BASE for Dos: A Business User's Guide, Condliffe
Macros, Menus and Miracles for Lotus 1-2-3, Lunsford
Desktop Publishing with Pagemaker for the IBM PC AT, PS/2, and Compatibles, Bove and Rhodes
Desktop Publishing with WordPerfect 5.0, Tevis
Using dBASE IV: Basics for Business, Brownstein
Using Lotus Agenda, Goodman
Using Supercalc 4: The New Generation, Doyle
Lotus 1-2-3 Release 2 A.S.A.P., Williams
Business Statistics Using Lotus 1-2-3, Kilpatrick
Turbo C Dos Utilities, Alonso
The Power of Quattro, Salkind
The 80386/387 Architecture, Morse
The 80286 Architecture, Morse

DOS Productivity Tips & Tricks

Gilbert Held

WILEY
John Wiley & Sons, Inc.

New York Chichester Brisbane Toronto Singapore

Publisher: Stephen Kippur
Editor: Therese A. Zak
Managing Editor: Ruth Greif
Assistant Managing Editor: Corinne McCormick
Editing, Design, and Production: Impressions, Inc.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought. FROM A DECLARATION OF PRINCIPLES JOINTLY ADOPTED BY A COMMITTEE OF THE AMERICAN BAR ASSOCIATION AND A COMMITTEE OF PUBLISHERS.

Copyright © 1989 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permission Department, John Wiley & Sons, Inc.

Library of Congress Cataloging-in-Publication Data

Held, Gilbert
  DOS productivity tips & tricks

ISBN 0-471-60895-5

Printed in the United States of America
89 90 10 9 8 7 6 5 4 3 2 1

Preface

The disk operating system (DOS) used with the IBM PC Series and compatible personal computers includes a large number of valuable functions and facilities. With these functions and facilities you can significantly increase the productivity of your computer operations. Exploring these DOS features is the focus of this book.

To permit the largest number of readers to benefit from the DOS tips and techniques presented, I divided it into two parts.

Section One is intended as a quick review of DOS concepts and characteristics. In addition to serving as a review for many readers, these chapters also include examples explaining concepts and characteristics that will be both of an immediate use and applicable for increasing your computer productivity. While more experienced readers may prefer to skim through or bypass the material presented in Section One, its presence as well as the comprehensive index included in this book should facilitate the review of DOS features and facilities when required by the reader.

Section Two focuses on developing specific productivity tips and techniques. Here you create and store sequences of DOS commands in batch files that automatically perform predefined functions. Such functions developed here with DOS commands fall into four main areas—file and subdirectory operations, screen control, printer control, and utilities. Over 20 distinct examples of DOS command sequences are included and fully described. Many of the functions performed by this book's productivity enhancement examples are marketed as standalone software programs. However, you do not have to purchase expensive software—save a significant amount of money by entering and using the DOS batch files of command sequences.

Acknowledgments

The development of a book is a time-consuming process that requires the cooperation and assistance of many persons. First and foremost, I must again thank my family for their understanding and patience while I hibernated in my computer room for many more weekends than I care to admit.

Once again, I would like to thank Mrs. Carol Ferrell for taking my draft manuscript and organizing it into a product that could be delivered to my publisher. Last, but not least, I would like to thank Teri Zak for providing me with the opportunity to put into print here the DOS tips and techniques I developed while presenting seminars in Europe that covered the use of the IBM PC family of computers.

Trademark Acknowledgments

COMPAQ is a registered trademark of COMPAQ Computer Corporation.

dBASE and MultiMate are registered trademarks of Ashton-Tate Corporation.

Epson LQ-1500 and RX-80 are trademarks of Epson America, Inc.

GW-BASIC, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation.

IBM, IBM PC, IBM PC XT, IBM PC AT, PC DOS, and Personal System/2 are registered trademarks of IBM Corporation.

Lotus 1-2-3 is a registered trademark of Lotus Development Corporation.

Safepark is a trademark of Prime Solutions Inc.

Contents

SECTION ONE: REVIEWING DOS

1 CONCEPTS AND CHARACTERISTICS *5*

Evolution of DOS *6*
DOS Components *8*
Prompts and Device Names *10*
File Specifications *11*
Global File Symbols *19*
Keyboard Control *20*
I/O Redirection *22*
Summary *29*

2 HIERARCHICAL DIRECTORY STRUCTURE OPERATIONS *31*

Purpose of Hierarchical Directories *32*
Directory Construction *33*
Summary *43*

3 BATCH FILE OPERATIONS *45*

Creating and Using Batch Files *46*
Batch Commands *49*
The AUTOEXEC.BAT File *60*

SECTION TWO: PRODUCTIVITY TIPS & TRICKS

4 FILE AND SUBDIRECTORY OPERATIONS *65*

Caveats for Batch Files *66*
Format Protection Schemes *67*
Enhancing Single-Drive Copying Operations *71*
Subdirectory and File Security *74*
Directory Mapping *76*
The File Locater *88*
Creating File Reference Commands *96*
Master Menus for Fixed Disk Systems *99*
Directory Navigation *121*

5 SCREEN CONTROL *125*

Screen and Keyboard Control *126*
Controlling a Color Display *128*
Examining Screen Control Codes *135*
Summary *137*

6 PRINTER CONTROL *139*

Printer Port Swapping *140*
Printer Spooling *142*
Label Generator *147*
Printer Mode Control *152*
Printer Port Testing *179*

7 UTILITY ROUTINES *183*

Creating DOS Command HELP Files *184*
A DOS Database *188*
Audio and Graphics Generation *194*

Disk Directory Label Generator *198*
Eliminating Illegal Subdirectory Names and Filenames *202*
Programming Function Keys and Keyboard Macros *203*
Performing Operations at Defined Times *210*
Text Search *213*
Creating a Calculator *218*
Computer Usage Logging *229*
Eliminating Disk Fragmentation *239*
Effective Use of RAM Disks *245*
Exit Code Processing *248*
Date and Time Access *252*
Determining the Size of a Directory *256*
Suppressing Messages *277*

INDEX *281*

DOS Productivity Tips & Tricks

SECTION ONE

Reviewing DOS

1

Concepts and Characteristics

This chapter reviews the key concepts and characteristics of DOS. Beginning with a short discussion of the evolution of the IBM PC's operating system, the chapter examines the components of the operating system, the method used to create device names, the components of a file specification, and the use of global or wildcard characters. Then, after reviewing the structure and function of DOS commands, the chapter concludes by investigating I/O redirection, pipes, and filters.

EVOLUTION OF DOS

During the development of its first personal computer, IBM negotiated agreements with several vendors for hardware and software. One such agreement was with the Microsoft Corporation, which in 1981 was a small firm specializing in BASIC language interpreters for personal computers. Microsoft had previously obtained an operating system known as 86-DOS from Seattle Computer Products. Under the agreement with IBM, Microsoft modified that operating system to support the IBM PC.

When the IBM PC was introduced, the operating system developed by Microsoft and marketed by IBM was known as PC DOS, an acronym for Personal Computer Disk Operating System. PC DOS is generally compatible with Microsoft's DOS, the latter better known as MS-DOS. Microsoft markets MS-DOS for use on IBM PC-compatible personal computers currently manufactured by more than 100 vendors. The major differences between the two versions of the operating system are in utility programs that access on-line storage devices, such as the DISKCOPY, DISKCOMP, and FORMAT programs. These utility programs normally are written to perform input/output (I/O) directly with the hardware and may be varied to correspond to the differences between IBM PCs and compatible systems. Thus, in some situations, one or more utility programs on an MS-DOS diskette may not work or may operate incorrectly if you use them on an IBM PC. Similarly, a PC DOS utility program may not work or may operate incorrectly when executed on a compatible computer.

Despite these minor differences, the operation and use of MS-DOS and PC DOS—as well as the tips and techniques presented in this book—are applicable to both IBM and compatible computers, using the operating system designed for your particular computer. Thus, this book uses the terms *MS-DOS* and *PC DOS* interchangeably, as well as the generic term *DOS,* which is often the simple term used by IBM and other personal computer vendors to reference the disk operating system.

Versions of DOS

The first release of DOS was Version 1.0, which was introduced when the IBM PC was announced in 1981. Since then, there have been two major revisions and approximately a half dozen minor revisions to the software.

The nomenclature used by Microsoft and IBM in labeling their versions of the operating system takes the form *X.YZ.* The *X* denotes the major release under which that version of DOS can be categorized. The *Y* indicates a substantial revision to the major release, whereas *Z* indicates a minor revision to the major release. This has become a popular, de facto standard for labeling many personal computer software products.

DOS Version 1.0 supported only single-sided diskette drives. This limitation was quickly corrected in Version 1.1, which supported double-sided disks with a capacity of 360 kilobytes (K bytes).

When the IBM PC XT was introduced in 1982, a hard disk (or fixed disk) was included as standard equipment. The 10 megabyte (M-byte) storage capacity of the hard disk was approximately 27 times the storage capacity of a 360K-byte diskette. This increase in storage capacity required the use of a hierarchical file structure for effective data management on that storage medium. Thus, accompanying the introduction of the IBM PC XT was a new major release of the operating system, DOS Version 2.0, which supported hierarchical file structures as well as the operation of a hard disk.

In 1984—three years after the introduction of the original PC—IBM announced the PC AT, together with DOS Version 3.0. This version of the operating system added support for the PC AT's high-capacity, 1.2M-byte diskette drive; the capability to change diskette and disk volume labels; support for networking applications; and a warning message issued by the FORMAT command, as well as several additional features that are discussed later in this book. Releases 3.1 and 3.2 followed shortly thereafter, adding additional support for networking applications and, under DOS 3.2, support for the 3½-inch diskette drives first introduced in the laptop IBM PC Convertible.

At the time this book was written, DOS Version 3.3 was the latest version of the disk operating system for the IBM PC family and compatible computers. This version of the operating system was released in April 1987, concurrently with the announcement of the Personal System/2 family of IBM personal computers. DOS Version 3.3 differed primarily from Version 3.2 in the addition of support for different media introduced for the PS/2 family, including the high-capacity, 1.44M-byte, 3½-inch diskette drives that are standard on many models in that series.

DOS COMPONENTS

DOS is actually a collection of utility programs that perform data management and peripheral device control functions. Depending upon your version of DOS, there may be as many as 60 functions, most of which are separate executable files stored on your DOS disk or in the DOS directory of your hard disk. They are invoked from the disk or directory when you enter an appropriate command. These commands and the files they reference are collectively referred to as *external commands,* because they do not reside in memory but on external on-line storage. Thus, you have to load them each time you use them.

In contrast to external commands, those commands loaded into memory when the operating system is booted or initialized are known as *internal commands.* When you enter an internal command name at the keyboard it is immediately executed, because the code for the command is already memory-resident.

Each time DOS is initialized, three files are loaded into memory. The first two files are called IBMBIO.COM and IBMDOS.COM in IBM PC DOS. These files are known as *hidden files,* because they are not listed when you list a directory of the disk on which the programs reside. In MS-DOS, these files are named BIO.COM and MSDOS.COM, respectively. The first file provides a standard interface to the hardware and supplements the Basic Input/Output System (BIOS) contained in the computer's read-only memory (ROM). The second file is responsible for interpreting commands issued by application programs and converting those commands into a form recognizable by the BIOS.

The third file, COMMAND.COM, is as its name implies a command interpreter. This is the only one of the three core DOS files that is shown in a directory listing. COMMAND.COM interprets keyboard entries and distinguishes between internal and external commands as well as invalid entries. When the command interpreter recognizes the entry of an internal command, it immediately executes that portion of the code in memory that corresponds to the command. Similarly, when the command interpreter recognizes the entry of an external command, it loads the program file that corresponds to the command.

PROMPTS AND DEVICE NAMES

Normally the diskette drive in the left device housing area is assigned the drive identifier A, followed by a colon (:), or A:. On systems such as some PC AT-compatibles that have only one device housing area, the diskette drive in the upper portion of that area is assigned the drive identifier A:.

If you insert a diskette containing a copy of the operating system in the computer's A drive, after power is turned on or when you perform a system reset by simultaneously pressing the CTRL, ALT and DEL keys, the DOS prompt A> is displayed. This prompt indicates that drive A is the default drive and any commands you enter without a preceding drive designator will reference that drive.

To change the prompt A> to a new default drive designator, simply type the new drive designator followed by a colon. Thus, the following annotated display illustrates how the DOS prompt (initially A) is changed to make drive B the default drive.

```
A> original prompt
A>B: new drive designator
B> new prompt
```

Normally, device names A and B are assigned to diskette drives, whereas device names C and D are assigned to hard disks.

FILE SPECIFICATIONS

You define every file on a diskette or hard disk with information that uniquely identifies that file. This unique information is called the *file specification* and can include the drive identifier, pathname, filename, and file extension. You must enter the filename, but the other three elements in the file specification are optional. If you don't specify them, DOS will assign default values to those elements as it works with the file.

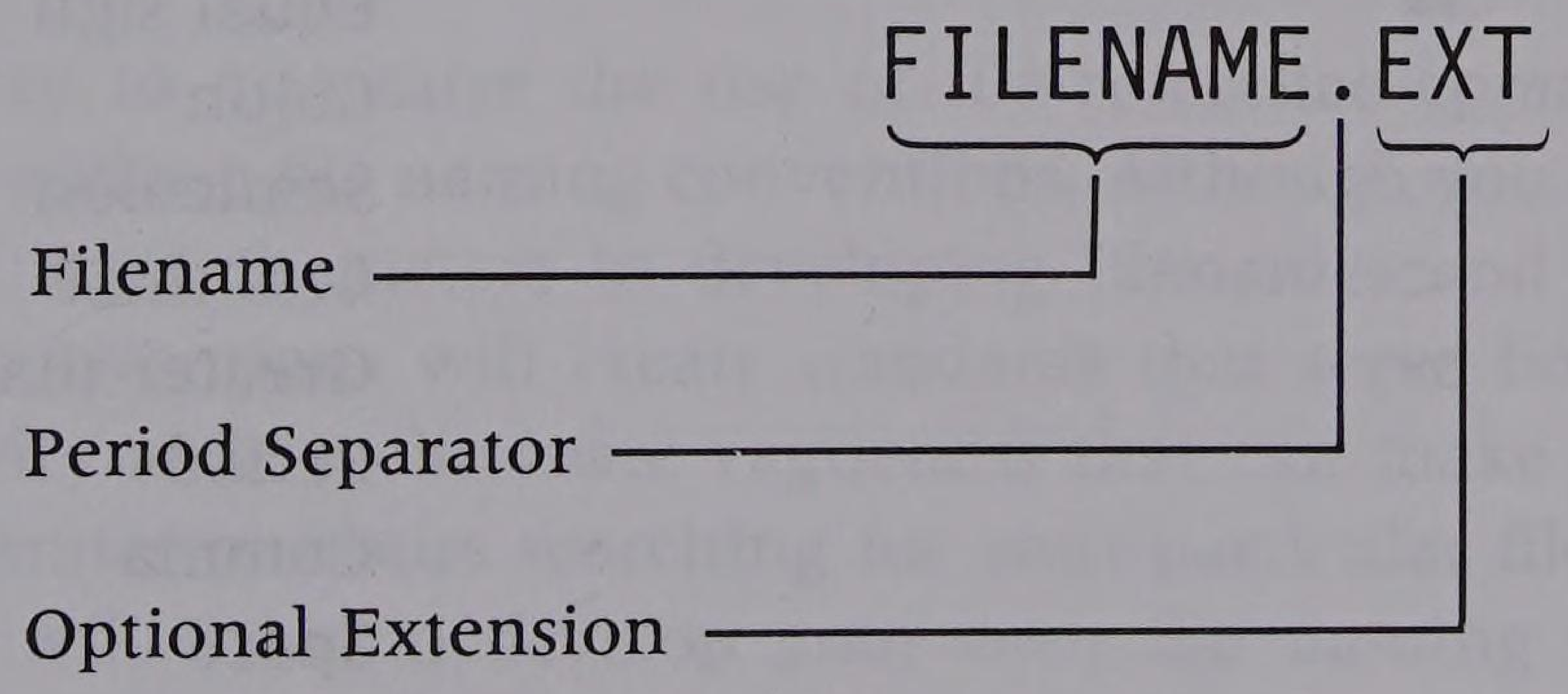
Filenames and Extensions

A *filename* consists of one to eight characters that can be used as a primary description of the information in the file. The file extension is an optional one to three additional characters following the filename and preceded by a period; it can further define the information contents of the file. Figure 1.1 illustrates the relationship of the filename and file extension.

Both uppercase and lowercase characters can be used in filenames and extensions because DOS does not distinguish between

Figure 1.1

Relationship of the Filename and File Extension



them. Depending upon your version of DOS, certain characters may not be legal in filenames and extensions. Table 1.1 lists the invalid characters under PC DOS Version 3.3, which represents the majority of characters that cannot be used in filenames and extensions for all versions of DOS.

As you assign filenames and extensions, there are certain names to avoid. These names are used by the operating system to reference specific components in the computer. Table 1.2 lists the DOS reserved names that cannot be used as filenames and the computer component or device they reference. Table 1.3 summarizes reserved extensions and their meaning or DOS usage. These extensions should only be

Table 1.1

Invalid Filename and Extension Characters

Character	Name
"	Quotation marks
/	Slash
\	Backslash
[	Left bracket
]	Right bracket
¦	Broken vertical bar
=	Equal sign
:	Colon
;	Semicolon
<	Less than
>	Greater than
.	Period
,	Comma
	Space

Table 1.2

DOS Reserved Names

Reserved Name	**Device**
CON:	Console keyboard/screen
AUX: or COM1:	First serial communications port
COM2:	Second serial communications port
COM3:	Third serial communications port
COM4:	Fourth serial communications port
LPT1: or PRN:	First parallel printer port
LPT2:	Second parallel printer port
LPT3:	Third parallel printer port
NUL:	Nonexistent device for use in application program testing

used when you are creating one of the indicated file types or manipulating a previously created file with a DOS command. Otherwise, substantial confusion over the type of file can result, or the file may not operate correctly.

File Naming Conventions

A key way to maximize the use of file reference commands is to establish and use file naming conventions. Although you can use any group of legal characters in developing filenames and extensions, naming conventions will create standards that serve both to boost productivity and to eliminate vagueness that can make other users spend minutes or hours searching for your particular files.

You will want to develop your own file naming system, depending on your individual or organizational requirements and the

Table 1.3

Reserved Extensions

Extension	Meaning
.BAK	Backup file
.BAT	DOS batch file
.CHK	Assigned to files recovered by CHKDSK
.COM	Program file directly executable by DOS
.EXE	Program file directly executable by DOS
.MAP	Default extension for list file created by DOS linker program
.OVL	Extension used by DOS for overlay files
.REC	Extension used by DOS for RECOVERed files
.SYS	Extension used by DOS for files containing system configuration and device drivers
.$$$	Extension used by DOS for temporary files

particular application. As an example, a file containing accounts payable information for 1988 might be named ACTPAY88.DAT, with the extension .DAT used to designate the file as a data file.

On the other hand, a large number of file extensions have been predefined and are accepted as de facto standards in addition to those extensions previously listed in Table 1.3. Table 1.4 gives, in alphabetical order, a list of de facto file extension standards.

Drive Identifier

The drive identifier is a letter, followed by a colon (:), which specifies a physical drive location. In some instances the drive identifier is optional and can be omitted. As an example, consider the file named

Table 1.4 De facto File Extension Usage

Extension	File Type
.ASM	Assembly language program in source code
.BAK	Backup file
.BAS	BASIC program
.BAT	Batch file containing DOS commands
.BIN	Binary file
.C	C language program in source code
.CHK	File recovered by CHKDSK
.COB	COBOL language program in source code
.COM	Command or program directly executable by DOS
.DAT	Data file
.DOC	Document file, usually created by word processor
.EXE	Executable relocatable program
.FOR	FORTRAN program in source code
.LIB	Library program
.MAC	Macro for assembly language program
.MAP	Link program listing
.OBJ	Machine language (object) version of compiled program
.OVL	Application program overlay file
.OVR	Compiler program overlay file
.PAS	Pascal language program in source code
.PIC	Screen (picture) display image
.REC	Recovered file
.REL	Relocatable object version of compiled program
.SYS	System configuration file
.TMP	Temporary file
.TXT	Text file
.$$$	Temporary file

ACTPAY88.DAT, which is located on drive C. If the default drive is not drive C, the drive identifier must precede the filename and extension to reference the file and its physical location. Thus, the file would be referenced as C:ACTPAY88.DAT. However, if drive C is the default drive, you could omit the drive identifier.

Paths

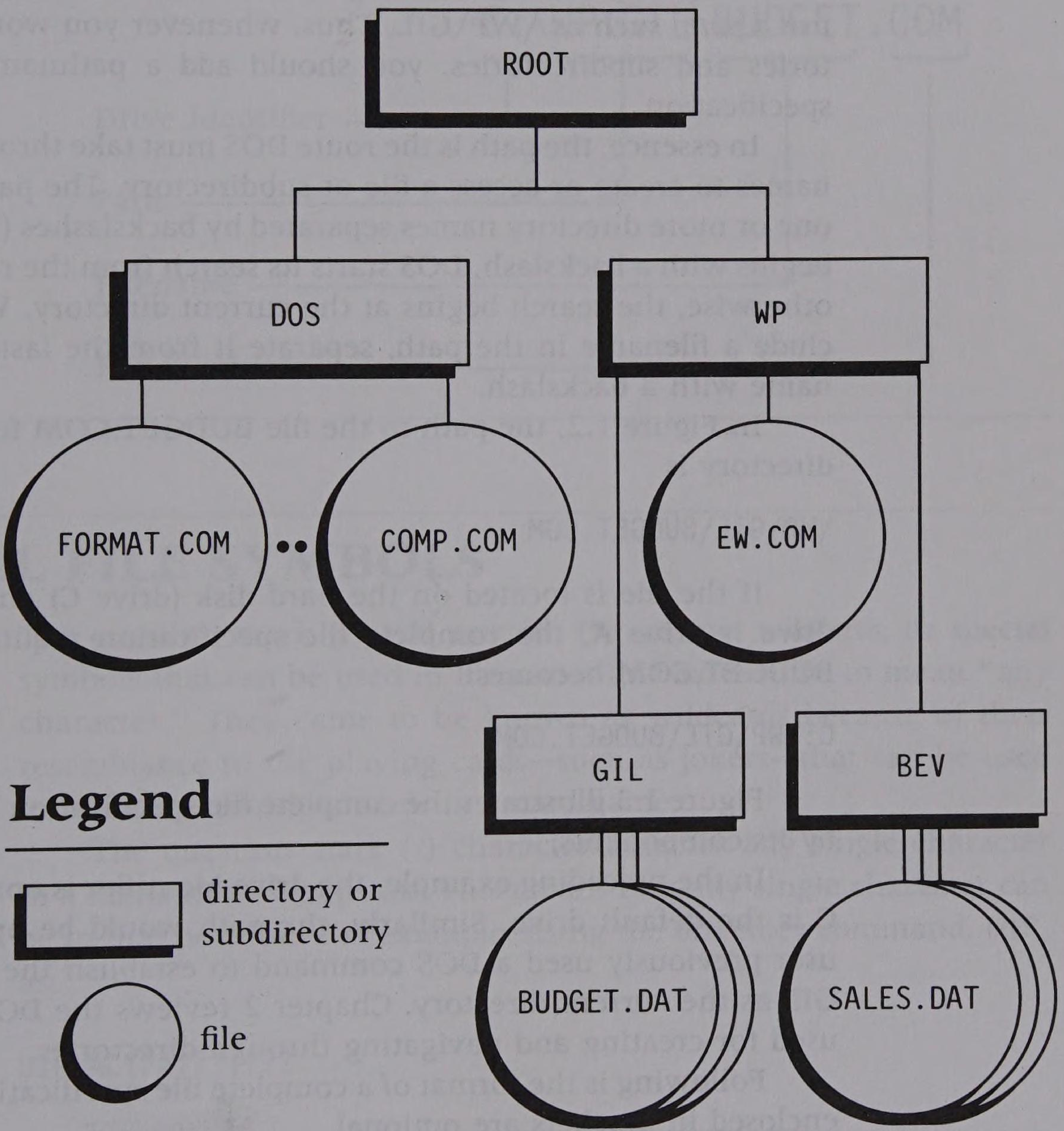
When IBM introduced large storage capacity (fixed disks) with the PC XT, a major revision to DOS was required to manage the complex storage. Although several major and minor revisions to DOS have occurred since Version 2.0, the hierarchical file structure of that version remains basically intact today for working with a large number of files.

Under the hierarchical file structure of DOS Version 2.0 and later revisions, a *root directory* is automatically created on each diskette and hard disk when the media are formatted. The root directory of a specific disk drive is indicated by a drive designator followed by a backslash. For example, A:\ is the root directory of a diskette in drive A, while C:\ is the root directory of the hard disk assigned to the device identifier C:. You can store a mixture of files and other directories in the root directory.

The other directories are commonly called subdirectories, because they are nested under the root directory. Figure 1.2 illustrates a directory structure that, although appropriate for a hard disk, could also be used on a diskette.

As you examine Figure 1.2, note that two subdirectories were established directly under the root directory—DOS and WP. Under the DOS subdirectory you might locate most of the DOS utility programs so they can be found easily when required. Under the WP subdirectory in this example, a word processing program named EW.COM and word processing files will be stored. If we assume that

Figure 1.2 **Sample Hierarchical Directory Structure**


ROOT
DOS
WP
FORMAT.COM
COMP.COM
EW.COM
GIL
BEV
BUDGET.DAT
SALES.DAT
Legend
directory or subdirectory
file

two persons use the word processor, it might be appropriate to separate their data files in different subdirectories, such as GIL and BEV.

To specify the route through the hierarchical file structure, you use a *path,* such as \WP\GIL. Thus, whenever you work with directories and subdirectories, you should add a pathname to the file specification.

In essence, the path is the route DOS must take through directory names to create or access a file or subdirectory. The path consists of one or more directory names separated by backslashes (\). If the path begins with a backslash, DOS starts its search from the root directory; otherwise, the search begins at the current directory. When you include a filename in the path, separate it from the last subdirectory name with a backslash.

In Figure 1.2, the path to the file BUDGET.COM from the route directory is

```
\WP\GIL\BUDGET.COM
```

If the file is located on the hard disk (drive C) and the default drive is drive A, the complete file specification required to access BUDGET.COM becomes:

```
C:\WP\GIL\BUDGET.COM
```

Figure 1.3 illustrates the complete file specification broken down by its components.

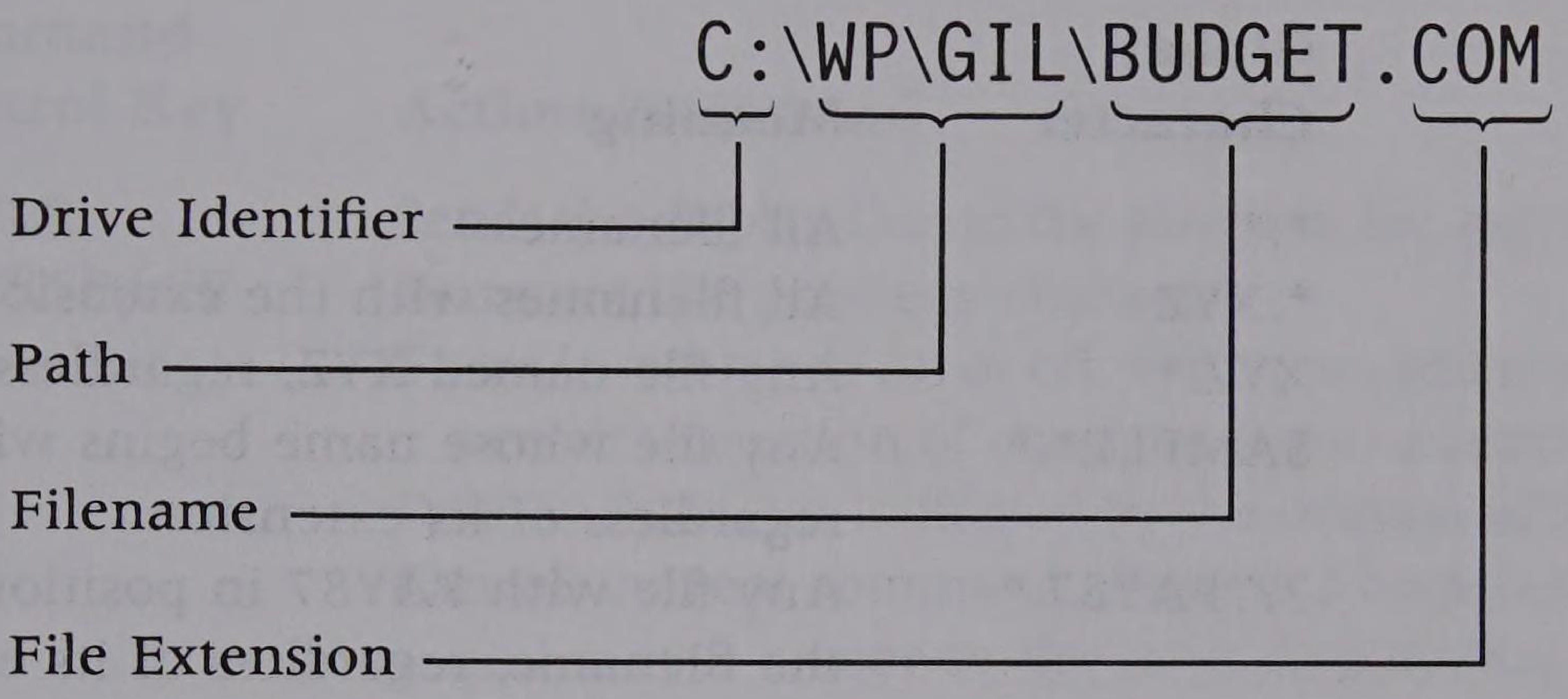
In the preceding example, the drive identifier is optional if drive C is the default drive. Similarly, the path would be optional if the user previously used a DOS command to establish the subdirectory GIL as the current directory. Chapter 2 reviews the DOS commands used for creating and navigating through directories.

Following is the format of a complete file specification; the items enclosed in brackets are optional.

```
[d:][pathname]filename[.ext]
```

Figure 1.3

Composition of a Typical File Specification



GLOBAL FILE SYMBOLS

The question mark (?) and asterisk (*) are two *wildcards,* or special symbols that can be used in filenames and extensions to mean "any character." They came to be known as wildcards because of their resemblance to the playing cards—such as jokers—that can be used to represent multiple cards in a card game.

The question mark (?) character matches any single character in a filename or extension. This means that any single character can be in that position. For example, using the directory command, DIR, as:

```
DIR ACTPAY??.???
```

causes any file in the directory that has ACTPAY in positions one through six of the filename to be displayed, regardless of its extension.

Table 1.5 Examples of Global Character Use

Global Character	Meaning
.	All filenames
*.XYZ	All filenames with the extension .XYZ
XYZ.*	Any file named XYZ, regardless of its extension
SAMPLE*.*	Any file whose name begins with SAMPLE, regardless of its extension
???PAY87.*	Any file with PAY87 in positions 4 through 8 of the filename, regardless of its extension

The asterisk (*) matches all characters from the asterisk's position to the end of the filename or extension. Thus, if you enter `DIR ACT*.*` all files in the directory that have ACT in positions one through three of the filename are displayed, regardless of their extensions. Table 1.5 lists five examples of the use of global characters and their meaning.

KEYBOARD CONTROL

Although you're probably quite familiar with the use of an IBM PC or compatible keyboard, several functions that DOS assigns to individual keys and key combinations warrant attention. Table 1.6 lists the keys that are applicable for editing commands and gives their actions. Of course, once you press the ENTER key the command is executed as indicated.

One of the more useful editing keys is the F3 function key. When you press this key, the entire previously issued command is copied

Table 1.6

DOS Command Control Keys

Command Control Key	Action Performed
ENTER	Sends the display line to the program for execution
BACKSPACE	Removes the previous character
INS	Turns insert mode on or off. When on, insert mode permits insertion of characters into a command
DEL	Deletes character indicated by the command buffer
ESC	Cancels current command, displays a backslash (\\), and moves cursor to the next display line
F1	Copies the next character from the command buffer
F2	Copies characters from the command buffer to the command line until the specified character is found
F3	Copies all remaining characters in the command buffer to the command line
F4	Moves over all characters in the command buffer until the specified character is reached
F5	Copies the command line to the command buffer without executing the command

to the command line, where it can be modified or used again as is. This key is extremely useful if, as an example, you want to copy the contents of several diskettes to a subdirectory on a fixed disk. The command

```
COPY A:*.* C:\WP\GIL
```

Keyboard Control

could easily be repeated by pressing F3 each time the contents of a diskette in drive A are to be copied to the indicated subdirectory on drive C. You would simply press the F3 key, then press ENTER. Unfortunately, because DOS does not display the contents of the command buffer as you perform editing operations, the other editing keys are of limited use. For this reason, many persons prefer to modify a command by typing it again.

Operational Control Keys

Several multiple-key sequences govern the control of DOS operations. Pressing CTRL-ALT-DEL to reboot the system is an example of a multiple-key sequence. These so-called operational control keys are usually enabled; however, certain programs may disable one or more of these multiple-key sequences to prevent an inadvertent keypress from affecting the operation of the program. Table 1.7 lists the multiple-key sequences that affect the control of DOS operations. It gives a brief description of the action for each key sequence.

I/O REDIRECTION

In its default mode of operation, DOS accepts input from your keyboard and directs the results of DOS commands to your monitor. These two devices are known as the *standard I/O devices* and, unless you specify otherwise, DOS commands use them to obtain and deliver data. Thus, when you enter DIR to list a directory, DOS first scans the keyboard to recognize the command. Then, DOS displays the results of executing the command on the screen. You can use the greater than (>) and less than (<) characters to redirect standard I/O. I/O redirection provides a powerful mechanism for developing

Table 1.7 **Key Sequences That Control DOS Operations**

Key Sequence	Action Performed
CTRL-C or CTRL-BREAK	Cancels current command or current command line
CTRL-ENTER CTRL-ENTER	Permits you to go to the next display line on the screen to continue entering the line you are typing
CTRL-P or CTRL-PRTSC	Echoes the data displayed on the screen to the printer. Press again to disable printing
CTRL-S or CTRL-NUMLOCK	Suspends the screen display. Press any key to resume
SHIFT-PRTSC	Prints out the contents of the screen
CTRL-ALT-DEL	Resets the system and restarts DOS

many productivity routines that are developed in Section Two of this book.

Input Redirection

You specify input redirection using the < symbol between a command and a reserved name or file specification. Thus, the format for input redirection is

$$\texttt{COMMAND<}\left\{\begin{array}{l}\texttt{reserved name}\\ \texttt{file specification}\end{array}\right\}$$

where the braces indicate that a choice of one of the enclosed items is to be made.

One of the more common uses of input redirection is to change the execution requirements of programs. As an example, consider a program named PAYROLL whose execution normally requires the user to sit at the terminal and type a series of keyboard entries as the program executes. With input redirection, the user could first use EDLIN (the text editor included on the DOS diskette) or a word processor to create an ASCII file containing the keyboard entries required to execute the program. Assuming that the name of the file containing the keyboard entries is JAN88.DAT, the command

```
PAYROLL<JAN88.DAT
```

executes the payroll program with input to the program taken from the file JAN88.DAT.

One of the major benefits of input redirection is automating the execution of programs. A secondary benefit that may be just as important for productivity is the ability to store program input data in files. Then, if there are minor changes to the input data between successive program executions, you can make required changes quickly with a word processor or text editor.

Output Redirection

Output redirection is similar to input redirection—both can be used with reserved names or file specifications. The format of output redirection is

$$\texttt{COMMAND}\left\{\begin{matrix} > \\ >> \end{matrix}\right\}\left\{\begin{matrix}\texttt{reserved name}\\ \texttt{file specification}\end{matrix}\right\}$$

When output is redirected to a file, DOS first checks to see if the file exists. If it exists, the output overwrites the file unless you include two greater than symbols (>>) in the output redirection format, which then causes DOS to *append* the requested output to the end of a previously created file. If the file does not exist, DOS creates and automatically saves the requested file.

To understand how output redirection can be useful, consider the following DOS command line.

```
DIR>LPT1:
```

When this command line is executed, DOS routes the directory listing to the first parallel printer port, providing a mechanism to obtain a hard copy of the directory automatically.

Recall the PAYROLL program discussed under input redirection. Suppose the output of the program directed to the screen contains `successful` or `error` messages for five modules. In addition, assume that the messages scroll off the screen as the next module executes. Because the creation and use of an input file to respond to the program's prompts might still require you to read each screen, the operation is not really automated. In this instance, you might consider combining the I/O redirection capabilities as shown here:

```
PAYROLL<JAN88.DAT>SCREEN.PIC
```

In this example the payroll program would receive input from the file JAN88.DAT and send screen output to the file SCREEN.PIC. Later, you could use the DOS TYPE command to examine the screen information previously generated by the program.

Pipes and Filters

Piping enables you to chain DOS commands and programs together and automatically redirect the standard I/O. This permits the screen output of one command to be used as the keyboard input to another command or program.

The special symbol ¦ denotes piping in the command line. The ¦ serves as a delimiter between the output of one command and the input to the second command. The second command—which accepts data from a standard input device, modifies the data, and then outputs the results to a standard output device—is commonly known as a *filter*. DOS Version 3.3 includes three filter commands—SORT, MORE, and FIND.

SORT Command

The SORT command reads data from a specified input device, sorts the data, then writes it onto the standard or a specified output device. The format of this command is

```
SORT[/R][/+n]
```

where the bracketed items indicate optional parameters. If you choose to include the /R switch, the sort is performed in reverse order, while the /+n switch causes the sort to start with column *n*. If you don't specify the second option, the sort or reverse sort starts in column 1.

The format of the piping of standard I/O is

```
                  ⎧ >   ⎫ ⎧ reserved name      ⎫
COMMAND ¦ COMMAND ⎨ > > ⎬ ⎨                    ⎬
                  ⎩ <   ⎭ ⎩ file specification ⎭
```

As an example of piping using the SORT command, consider the following command line entry.

```
DIR C:¦SORT>LPT1:
```

In this example, the video output that normally results from the DIR command is first piped into the SORT command, which sorts the directory and then directs the sorted output to the first parallel printer port.

MORE Command

The MORE command is a filter that displays one screen of 24 lines of data and the message `-More-`, then pauses. When you press any key, the next screen is displayed. As an example of the use of the MORE filter, consider the following command line entry.

```
MORE<SORT/R<MEMO.DAT
```

The preceding command line sorts the file MEMO.DAT in reverse order and then displays it one screen at a time. Note that the SORT command sorts data according to the ASCII values of the characters it encounters. This means that when you use MORE with a data file, the file must be in ASCII format. In addition, because uppercase letters have a lower ASCII value than lowercase letters, the results of a sort operation can be unexpected. For example, PREPARATION will appear before Parameter. For this reason, good sort utilities (which the DOS SORT command is *not*) internally convert the sort keys to a single case before comparing them, so that the sorted output is in alphabetical order. SORT is also slower than a sick tortoise—don't try to sort more than 50 items with it unless your patience is infinite!

FIND Command

The third filter in DOS is the FIND command. This command can be used to search an ASCII file for the occurrence or nonoccurrence of a string included in the command. The format of the command is

```
FIND[/V][/C][/N]"string"[file specification]
```

If you include the /V switch, FIND displays all lines that do *not* contain the specified string. When you use the /C switch, FIND counts the number of lines that contain the specified string, while the /N option includes a line number for each displayed line.

The FIND command provides a valuable mechanism for constructing command sequences to perform database-related operations. To understand the usefulness of this command, assume you have an ASCII file named PHONE.DAT whose record format is illustrated in Figure 1.4.

Suppose you created and maintained the file with a word processing program at the central office of an organization. Other employees might want to obtain a telephone directory of staff in their departments. To do this, the employees would use the FIND filter to locate all records in the file that contain the string associated with their department code. If the department code were A106, all the

Figure 1.4

PHONE.DAT Record Format

1 12	13 23	24	25 27	28 36	37 40
Last Name	First Name	Ini- tial	Area Code	Telephone Number	Department Code

records containing that string could be printed by entering the following command line.

```
FIND "A106" PHONE.DAT>LPT1:
```

If you wanted a directory listing based upon the department code for the entire organization, you could use the SORT command with the FIND command in one line to accomplish this task. Because the department code begins in column 37 of each record, you must signify this in the SORT command by using the /+n option, with *n* assigned the value 37. Thus, you could use the FIND and SORT commands, piping the output of FIND to SORT and directing the output of the SORT command to the printer as shown:

```
FIND "A106" PHONE.DAT ¦ SORT/+37>LPT1:
```

SUMMARY

Now that we have reviewed the key concepts and characteristics of DOS, you can use this foundation of information to examine hierarchical directory structure operations. The use of DOS commands to create, modify, and delete subdirectories—as well as to navigate hierarchical directory structure—is presented in Chapter 2.

2

Hierarchical Directory Structure Operations

This chapter focuses on those DOS commands used in hierarchical directory structure operations. Commands in this category can create and delete directories and subdirectories, navigate through directories, and check the structure of directories.

PURPOSE OF HIERARCHICAL DIRECTORIES

Hierarchical directories were developed to enhance the management of files on hard disks, as well as to enable you to store more files on diskettes. Without a hierarchical directory structure, the design of DOS restricted the number of files that could be stored on a double-sided diskette to 112, even if room existed on the diskette to store additional programs or data files. In a hierarchical directory structure, each *subdirectory*—the term used for a directory entry—denotes a unique place on the storage medium. Due to this, files located in different subdirectories on the same disk are considered as separate entities from one another, even if the files have the same name. Because DOS manages files by their directory location, each subdirectory can be considered equivalent to a diskette or hard disk, permitting many more files to be stored and managed by the operating system.

With each subdirectory treated as a distinct location by the operating system, it is easier to manage files using a hierarchical directory structure. For example, consider a person who uses three specific application programs. In a hierarchical directory structure, the person could locate each application program and other files used with each program in specific subdirectories. Then, the user could list the programs and data files contained in each subdirectory with appropriate DIR commands. Without a hierarchical directory structure, when a user entered a DIR command hundreds or thousands of files representing all the programs and data files stored on a disk would be displayed.

DIRECTORY CONSTRUCTION

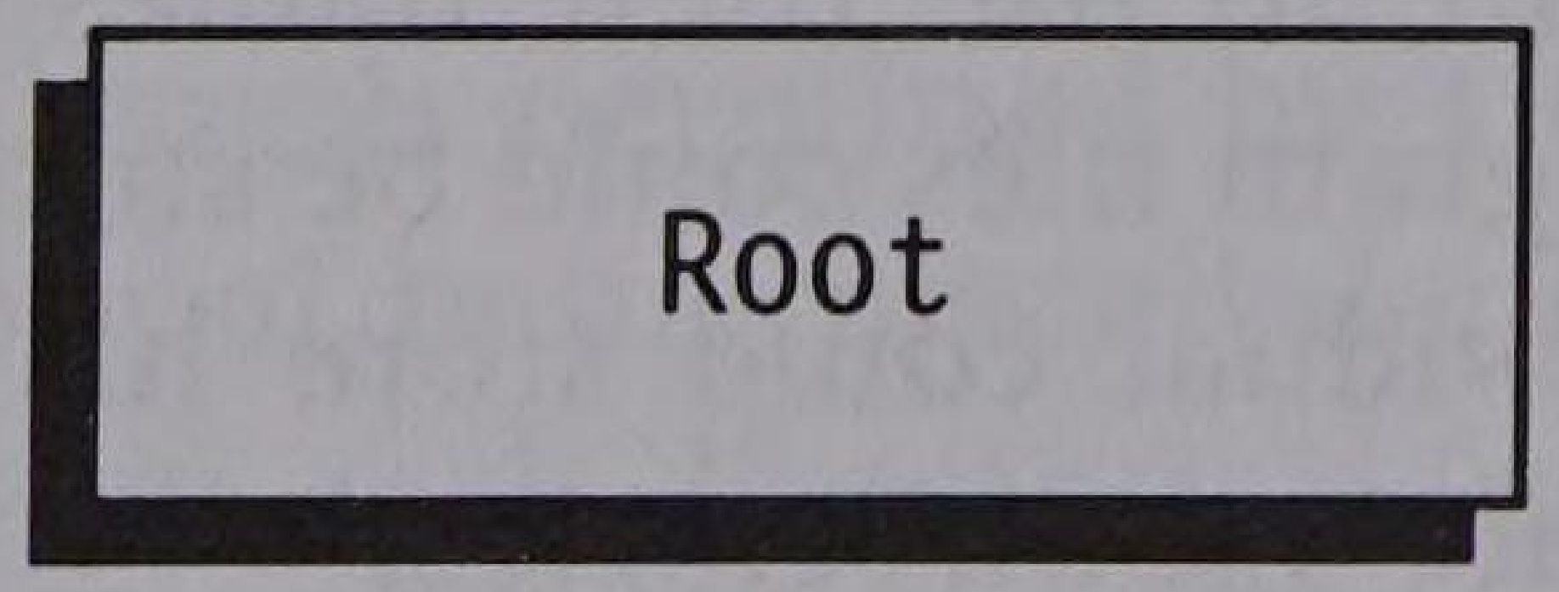
One of the advantages of a hierarchical directory structure is that it can be easily tailored to specific user requirements. This flexibility is contingent upon observing a few constraints that govern naming directories and nesting subdirectories. As we examine the construction of directories in this section, I will note when constraints apply.

Under DOS, when a diskette or fixed disk is formatted, access to all storage on the medium is initially assigned to one directory, known as the *root directory.* This directory is always located at the top of the hierarchical directory structure. It earns its name, *root,* from the fact that the initial path to all subdirectories starts in this directory. In the following examination of directory construction, I use rectangles to denote directories and circles for files in a series of illustrations.

In Figure 2.1 the initial file structure resulting from formatting a disk is illustrated. As indicated, only one directory exists—the root directory.

From the root directory, path names must be specified to provide the route through the directory structure required to access a specific directory or file. Path names start at the root and consist of strings of directory names separated from one another by the backslash (\) character.

Figure 2.1 **Initial Disk File Structure**



MKDIR Command

Directories are created through the MKDIR command; its format is

```
{MD   }
{MKDIR}[PATH]Directory name
```

Like many other directory-related commands, this command has an abbreviation. Thus, you can enter either MD or MKDIR, followed by the path and the name of the directory to be created. Assuming you are located at the root directory level and desire to create two directories named LOTUS and DBASE under the root, you would enter the commands

```
MKDIR\LOTUS
MKDIR\DBASE
```

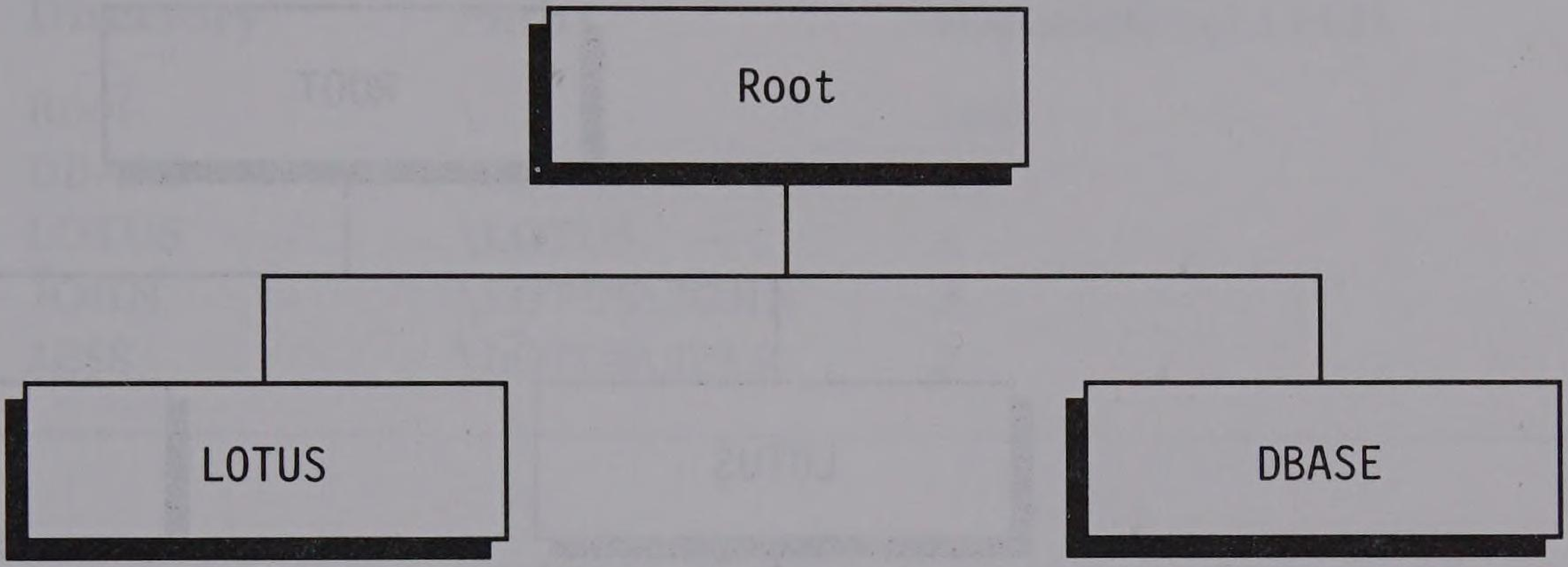
A hierarchical diagram illustrating the results of entering the two MKDIR commands is illustrated in Figure 2.2. As indicated, each of the newly created subdirectories is located at the same hierarchical level under the root directory.

Now that two subdirectories have been created under the root directory, assume that the LOTUS directory in which that popular program's data files are stored is to be shared by two persons. Two subdirectories could be established under the LOTUS directory. If John and Jess are the two persons who use LOTUS, it might be appropriate to use their names as subdirectories. Then, the spreadsheet program files could be stored under the LOTUS directory, while each individual could store his or her data files within his or her appropriate subdirectories.

If you are presently located at the root directory, you can create these two subdirectories by entering the following commands.

Figure 2.2

Result of Adding Subdirectories



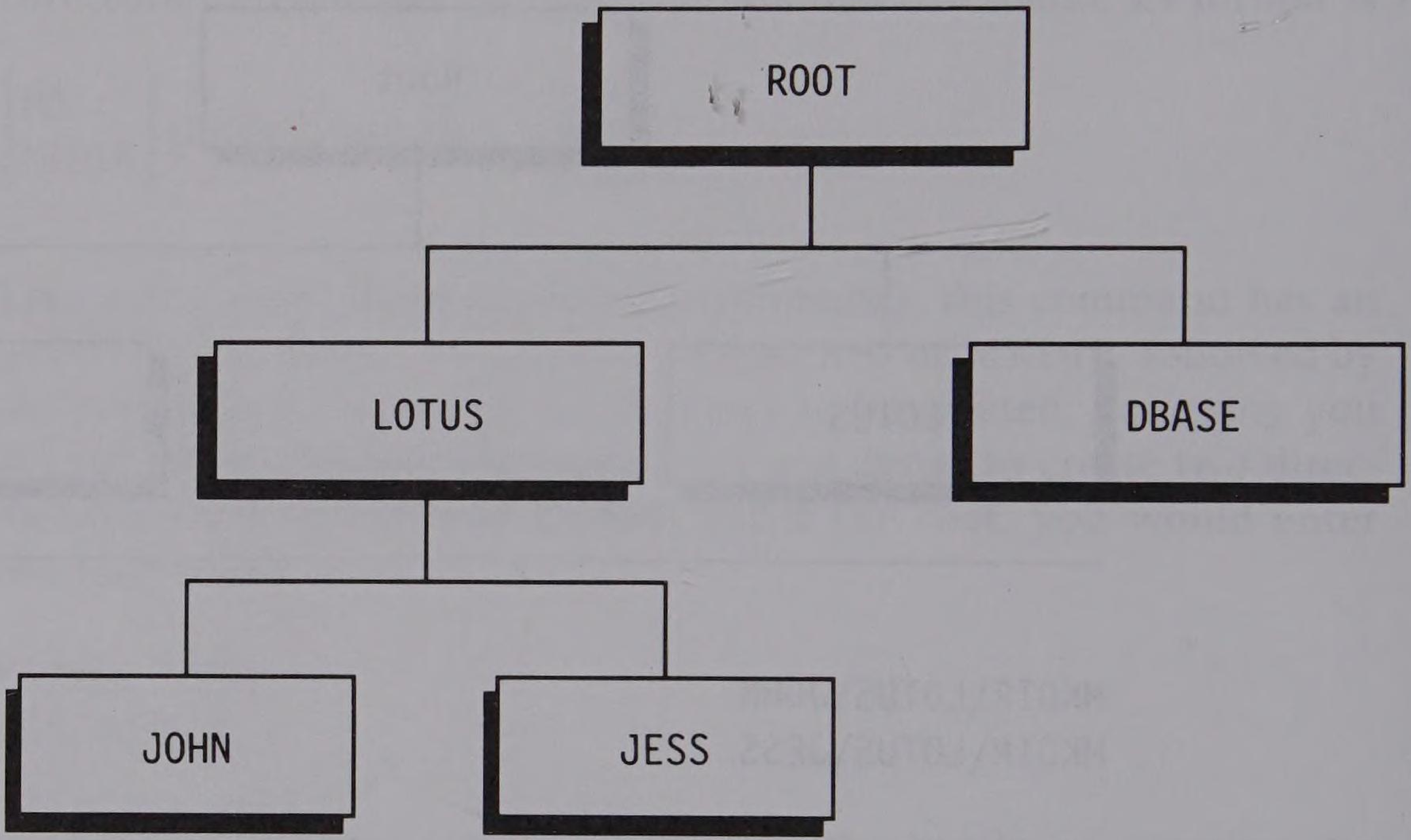
```
MKDIR\LOTUS\JOHN
MKDIR\LOTUS\JESS
```

As a result of the two new MKDIR commands, the LOTUS directory contains two nested subdirectories named JOHN and JESS. Figure 2.3 shows this structure. At this point, your hierarchical structure is as indicated in Table 2.1.

Constraints to Consider

If a requirement existed, you could continue to nest directories as long as the maximum path length did not exceed 63 characters. This is the maximum path length permitted by DOS; the actual number of subdirectories capable of being nested depends upon the number of characters used to form each directory name. Subdirectory names follow the same conventions as standard DOS filenames. That is, they can be up to eight characters in length and can have an optional

Figure 2.3 **Modified Directory Structure**



one- to three-character extension; a period (.) separates the extension from the directory name.

CHDIR Command

Although paths can be specified to appropriate subdirectories each time you have to work with a file or set of files, in many instances changing your current directory can facilitate DOS operations. The reason is that references to program and data file names in the current directory do not require the inclusion of path information. To better understand this concept, assume you are at the root directory in

Table 2.1 **Hierarchical File Structure of Figure 2.3**

Directory	Path	Hierarchical Level
Root	\	root
DBASE	\DBASE	1
LOTUS	\LOTUS	1
JOHN	\LOTUS\JOHN	2
JESS	\LOTUS\JESS	2

Figure 2.3. If the illustrated directory is located on drive C and you wished to copy the contents of a floppy diskette in drive A to the subdirectory JOHN, you could do so by entering the following command:

```
COPY A:*.* C:\LOTUS\JOHN
```

To use a second method to transfer the contents of the diskette to the subdirectory JOHN without specifying a path, you could first change the current directory on drive C. This is accomplished by the use of the change directory command, CHDIR or CD, whose format is:

```
⎧CHDIR⎫
⎨     ⎬[Path]Directory name
⎩CD   ⎭
```

To move down the hierarchical file structure to the subdirectory JOHN from the root directory you would issue the command:

```
CHDIR \LOTUS\JOHN
```

Directory Construction

Then, because the current directory is JOHN, you could enter the following DOS command line to copy the contents of the diskette in drive A to the subdirectory JOHN located on drive C.

```
COPY A:*.* C:
```

Note that by changing the current directory to JOHN you do not have to specify a path when you copy the contents of the diskette to the intended subdirectory—this is because the current directory or subdirectory is the default for copy procedures. While the two-level subdirectory hierarchy does not provide a significant savings in typing, if there are several operations to be performed in a deeply nested directory, it is normally easier to change the current directory to the intended subdirectory than to continue to specify long path names.

Regardless of the method used, after you copy the contents of the diskette to the subdirectory JOHN, you can use either procedure to copy JESS's files to her subdirectory and the files for LOTUS to its directory. Once this is accomplished, the directory structure appears as illustrated in Figure 2.4.

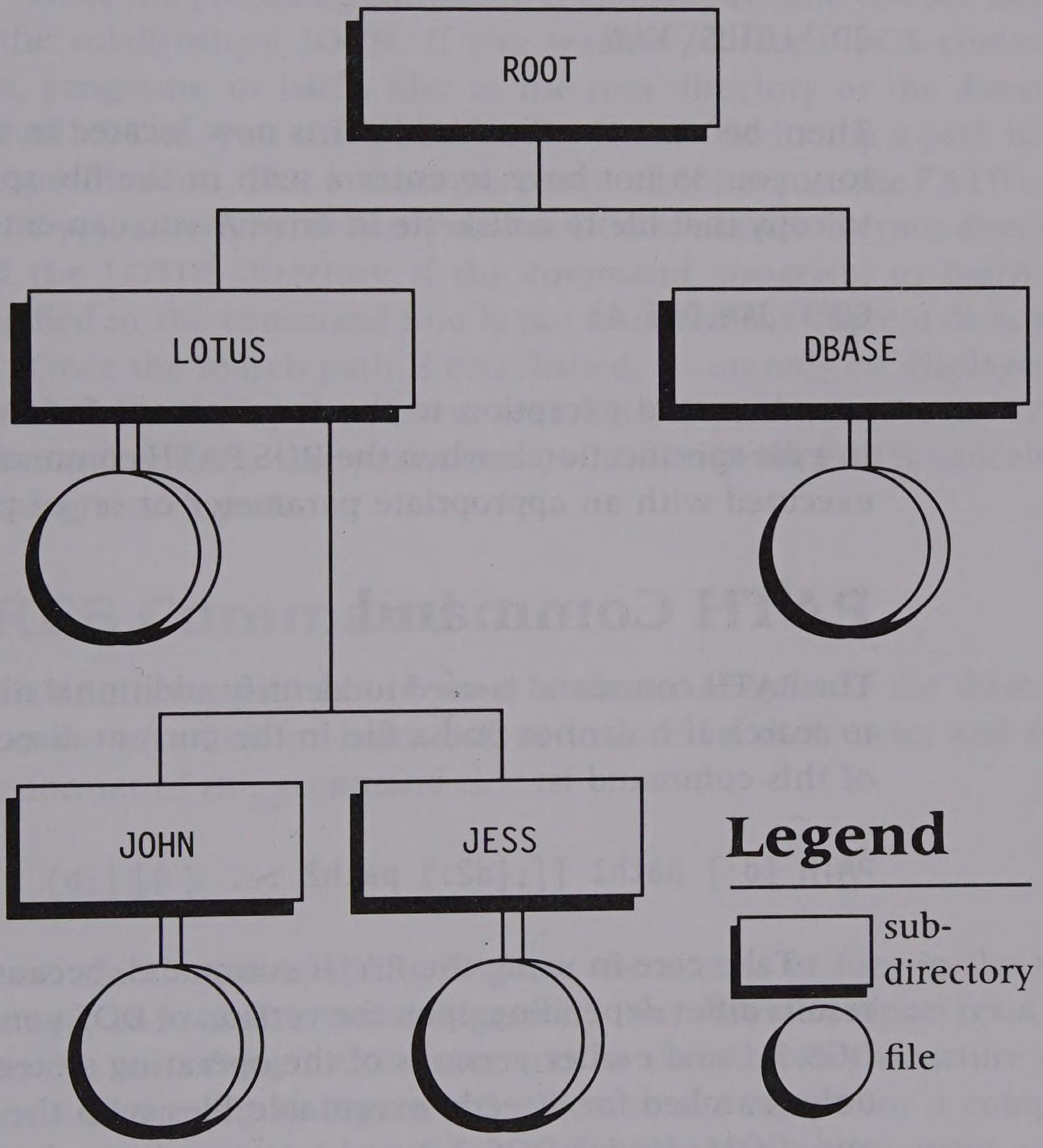
File References

In a hierarchical file structure, you must normally include the path in the file specification. One exception to this is when the file resides in the default or current directory, in which case the path can be omitted, as we just saw. All file references without a path name are assumed to reference the current directory. Thus, if you are at the root directory in the situation shown in Figure 2.4 and have a file named JAN.DAT in the directory named JOHN you wish to copy to a diskette in drive A, you could enter the command

```
COPY \LOTUS\JOHN\JAN.DAT A:
```

Figure 2.4

Directory Structure After Files Copied


ROOT
LOTUS
DBASE
JOHN
JESS
Legend
sub-directory
file

As an alternative, assume you previously made JOHN the current directory by entering the DOS command:

```
CD \LOTUS\JOHN
```

Then, because the file JAN.DAT is now located in the current directory you do not have to enter a path in the file specification. Now, to copy that file to a diskette in drive A you can enter the command:

```
COPY JAN.DAT A:
```

A second exception to the requirement for including the path in a file specification is when the DOS PATH command was previously executed with an appropriate parameter or set of parameters.

PATH Command

The PATH command is used to identify additional directories for DOS to search if it cannot find a file in the current directory. The format of this command is:

```
PATH [d:] path1 [[;[d2:] path2 . . . ]]
```

Take care in using the PATH command, because its operational results differ depending upon the version of DOS you are using. Under DOS 3.1 and earlier versions of the operating system, this command only searched for directly executable files with the extensions .EXE and .COM. Under DOS 3.2 and later versions, the PATH command causes DOS to search for a command, program, or batch file in other directories if they are not found in the current directory.

As an example of the PATH command, consider the following command line entry.

```
PATH \;\LOTUS
```

Once the preceding command is entered, assume you are located in the subdirectory JOHN. If you wanted to access DOS command files, programs, or batch files in the root directory or the directory named LOTUS, you could now do so without including a path in the file specification. This omission is permissible because the PATH command you entered previously causes DOS to search the root directory and the LOTUS directory if the command, program, or batch file specified in the command line is not found in the current directory.

Once the search path is established, it can only be displayed or canceled. To display the current search path, you can enter the PATH command without any parameters, while entering PATH cancels an existing search path.

TREE Command

The TREE command can be used to obtain a report of the directory structure of a disk or hard disk to include its directories and files. The format of this command is

```
TREE [d:][/F]
```

where the `/F` switch option displays the names of files in the root directory and subdirectories. Figure 2.5 illustrates the report resulting from the use of the TREE command on the directory structure previously illustrated in Figure 2.3. Because the report for a complex directory structure can require several minutes of time to print, upon occasion you may wish to consider directing the output to a file. Then, with techniques that will be described in Section Two of this book, you can process files created by the use of the TREE command to create customized reports.

Figure 2.5 Report Generated by TREE Command

```
DIRECTORY PATH LISTING
Path: \LOTUS
Sub-directories:  JOHN
                  JESS

Path: \LOTUS\JOHN
Sub-directories:  None

Path: \LOTUS\JESS
Sub-directories:  None

Path: \DBASE
Sub-directories:  None
```

RMDIR Command

Follow two rules when you want to remove a directory. First, remove all entries contained in the directory, including subdirectories and files. Second, the directory to be removed cannot be the current directory.

The format of the remove directory command is

```
{RMDIR}
{RD   }[Path]Directory name
```

Suppose you want to remove the subdirectory DBASE illustrated in Figure 2.3 and your current directory is the root directory. If there are no files or subdirectories under the DBASE directory, you can simply enter the command `RMDIR\DBASE`. Otherwise, you would have to remove any files or subdirectories before you could remove the DBASE directory.

SUMMARY

In this chapter we examined the purpose of hierarchical directory structures. Then we examined the use of several DOS commands that enable us to create and delete subdirectories, change directories, and copy files to different subdirectories. Now you can build upon this information to develop batch files—the subject of Chapter 3.

3

Batch File Operations

Batch files provide DOS users with the ability both to automate operations and to significantly boost the productivity of DOS and its users. With batch files you can customize DOS for other computer users, simplify complex systems, and perform other operations to simplify the use of a personal computer.

A batch file consists of a series of DOS commands that are executed when a single command referencing the name of the file is entered. Normally you test the operation of a batch file prior to using it regularly or providing it to others, and this ''debugging'' phase helps to eliminate many common causes of DOS operational problems. Such problems can include typing errors in complex DOS command lines, performing unintentional DOS operations, or lacking the ability or training necessary to perform DOS-related operations.

This chapter first examines how to create and use batch files. Then, using this information as a base, we will examine the operation of specific DOS batch commands.

CREATING AND USING BATCH FILES

A batch file consists of ASCII text that represents one or more DOS commands. Each command occupies a separate line in the file. To identify the file as a batch file, its extension must be .BAT. However, to execute the batch file at the DOS prompt, you can enter the batch file's name without its extension. When you do, DOS searches the current directory for a .COM, .EXE, or .BAT file with a name that matches the name you entered. You should therefore avoid assigning a name to a batch file that duplicates the name of a DOS command or another executable (.COM or .EXE) file.

Because a batch file is restricted to using ASCII text, it must be created by a process that generates that type of data. One of the most common methods of creating a batch file is to use the DOS COPY command to copy data from the console to your file. As an illustration of using COPY, consider the following example, which creates the batch file named WP.BAT.

```
A>COPY CON: WP.BAT
CD \WP
WRITE
CD \
^Z
```

In this example you must enter a CTRL-Z (^Z) by simultaneously pressing the CTRL and Z keys to terminate the COPY operation. Once you do so, the data entered from the console will be written onto the file WP.BAT. Then, by simply entering the command WP you can execute three separate command line entries.

The first line entry in the WP.BAT file changes the directory to WP. The next line entry, WRITE, is assumed to be a command that runs a word processing program with that name. Finally, the third line entry changes the current directory to the root directory. Although this is a simple example of a batch file, it illustrates how you can create a program that, while transparent to the user, navigates through a hierarchical directory structure to execute a program and then returns to the root directory once the use of that program is terminated. Thus, batch files can be extremely useful in simplifying operations for users who are not well versed in using the operating system.

Instead of using the COPY command to create batch files, you can create them with the help of the EDLIN text editor included with DOS. Most word processors can also store unformatted (ASCII) files. If you use a word processor, make sure that the resulting data is written onto disk as an ASCII file and not as the word processor's normal binary file or as an encoded text file. Otherwise, DOS will not be able to correctly interpret the information contained in the file.

Replaceable Parameters

One of the most helpful features of a batch file is its ability to use replaceable parameters. DOS permits up to 10 replaceable parameters to be used at a time within a batch file. These parameters are replaced by data the user supplies in the DOS command line when the batch file is executed.

The replaceable parameters that can be specified must be labeled %0 through %9, with %0 always replaced by a drive specifier, if required, and the filename of the batch file. As an example of the use of replaceable parameters, consider the following batch file.

```
A>COPY CON: WRITE.BAT
CD \WRITE
COPY %1.DAT %1.BAK
WRITE %1
```

Suppose the previously created batch file was executed by entering the DOS command line:

```
WRITE JANEMEMO
```

This would cause the batch program to automatically make a copy of a previously created word processing file named JANEMEMO before the user modifies the file. Thus, if you modify and save JANEMEMO.DAT you have the original stored as JANEMEMO.BAK (the backup file).

Here JANEMEMO is substituted for the replaceable parameter %1. Thus, after the directory is changed to WRITE, the batch program substitutes JANEMEMO for %1, so the file JANEMEMO.DAT is copied to the "backup" file JANEMEMO.BAK. Then, the batch program automatically executes a word processing program named WRITE, using the file JANEMEMO.DAT as input.

If the COPY command in the previously illustrated batch file was changed to COPY %1.DAT %2.BAK, two filenames would be required in the DOS command line. In this instance, entering WRITE JANEMEMO JANEBKUP would result in the sequential substitution of JANEMEMO for %1 and JANEBKUP for %2.

BATCH COMMANDS

When DOS Version 1.0 was introduced, it included only two commands designed specifically for use in batch files—REM and PAUSE. DOS 2.0 added five additional batch commands—ECHO, FOR, GOTO, IF, and SHIFT, as well as the undocumented ability to include comments by prefixing line entries with a period (.). This undocumented capability is also known as the *dot command.*

When DOS Version 3.0 was released, the batch commands remained essentially unchanged; however, the undocumented dot command was eliminated. The only major change to batch commands occurred with the introduction of DOS Version 3.3, which added the CALL command. Through the use of this batch command you can program one batch file to execute another batch file, similar to a subroutine call in conventional programming. Then, after the CALLed batch file is executed, control is passed back to the original batch file. This process is similar to the manner in which a subroutine returns control to a main program.

Table 3.1 indicates the batch commands supported by different versions of DOS. This table can be used to determine if you should obtain a more recent version of the operating system to take advan-

Table 3.1 DOS Batch Command Support

BATCH COMMAND	DOS Support 1.X	2.X	3.0	3.3
REM	yes	yes	yes	yes
ECHO	no	yes	yes	yes
PAUSE	yes	yes	yes	yes
FOR	no	yes	yes	yes
GOTO	no	yes	yes	yes
IF	no	yes	yes	yes
SHIFT	no	yes	yes	yes
CALL	no	no	no	yes

tage of one or more of the techniques presented in Section Two of this book.

REM Command

The REM command, which as in the BASIC language is an abbreviation for REMark, provides the capability to document your batch files when you include comments at appropriate locations in a file. The format of this command—which is only used in batch files—is

```
REM [remark]
```

As an example of the use of this command, let us modify our previously created WP.BAT file. The following example illustrates how you might identify each line in the batch file that performs an operation.

```
REM Change to WP directory
CD \WP
REM Execute word processing program
WRITE
REM Return to root directory
CD \
```

For clarity, many persons prefer to group REM commands together. Thus, as an alternative the file might be created using REM commands as follows:

```
REM Change to WP directory
REM Execute word processing program
REM Return to root directory
CD \WP
WRITE
CD \
```

The DOS commands used in our sample three-line batch file are essentially self-explanatory and for many persons probably do not require documentation. However, remarks in a complex batch file can save hours of future effort if you have to modify the file. Be considerate of people who might inherit your batch files—provide explicit documentation of the complex statements.

In place of the REM command, with DOS Version 2.X you can place a period (.) in the first column of lines you wish to use for comments. This undocumented feature unfortunately was eliminated when DOS Version 3.0 was released. Thus, unless you are using DOS 2.X and do not intend to upgrade to a more recent release, you should probably avoid using this undocumented feature.

ECHO Command

This command can enable or disable the display of DOS commands as they are executed from within a batch file. The format of this command is

$$\texttt{ECHO}\left[\begin{Bmatrix}\texttt{ON}\\ \texttt{OFF}\end{Bmatrix}\right]\texttt{[message]}$$

When used with no parameters, the command displays the current ECHO state:

```
A>ECHO
Echo is ON
```

ECHO ON is the default state when your system is powered on or you reset the system. The ECHO OFF command instructs DOS to inhibit the display of the DOS prompt and commands on the screen as they are executed from within a batch file. Similarly, an ECHO ON command causes the display of the DOS prompt and commands on the screen as they are executed from within a batch file.

If you include a message in the ECHO command, it will be displayed regardless of the ECHO state.

PAUSE Command

The PAUSE command can be used to suspend the execution of a batch file temporarily. The format of this command is

```
PAUSE [remark]
```

When the PAUSE command is executed, it first displays any remark included in the command line. Then, the command causes the temporary suspension of the execution of the batch file it is included in and displays the message `Strike a key when ready...` on a new line. Once a key is pressed, the batch file resumes execution at the line following the line on which the PAUSE command was located. The following short batch program illustrates how the PAUSE command can be used to provide information to the user. Note that FORMAT is a dangerous command to play with unless you absolutely know what you are doing!

```
COPY CON: FORMAT.BAT
ECHO OFF
XFORMAT A:
```

Assuming we previously renamed the DOS FORMAT file as XFORMAT and named our batch file FORMAT, the result of entering our new FORMAT command is

```
C>FORMAT
Insert new diskette for drive A:
and strike ENTER when ready
```

Note that the preceding batch file changed the use of the DOS FORMAT command to restrict its usage to formatting diskettes in drive A. Section Two examines in detail several methods you can use to prevent the inadvertent formatting of the hard disk.

FOR Command

The FOR command enables you to execute a specified DOS command iteratively. The format of this command is

```
FOR %%variable IN(file set)DO command
```

When the FOR command is executed, a single-letter variable included in the command is set sequentially to each member of the file set. After the variable is set to a member of the file set, the command following the DO specification is then executed. This process repeats until all file set members are processed. As an example, assume you wished to obtain a listing of all the batch files and data files under the subdirectory DBASE. You could create a batch file like this one:

```
COPY CON: LSTDBASE.BAT
CD \DBASE
FOR %%X IN(*.BAT *.DAT)DO TYPE %%X>LPT1:
CD\
^Z
```

Once the batch file named LSTDBASE is created, you would simply type its filename to change the directory to DBASE. Then, the FOR command assigns the variable X to the file *.BAT, which is actually a potential set of files, because the asterisk wildcard character is included in the file specification. Next, the TYPE command is executed for all files whose extension is .BAT, after which the variable X is assigned to the file set *.DAT and all files whose extension is .DAT are printed. As indicated in this example, you can include wildcard characters in the FOR command to increase the power and utility of this batch command.

GOTO Command

The GOTO command gives batch files the ability to perform branching operations. This command, as we will soon see, can be used with any combination of commands or conditional operators.

The format of the GOTO command is

```
GOTO [:]label
```

Here, the label is required to indicate a destination point in the batch file to which the program should branch.

The label is a string of up to eight characters, which must be prefixed by a colon (:) to indicate a branch entry point. When used in this manner, the label is contained on a separate line in the batch file. The following example illustrates the use of the GOTO command and the use of a label in a batch file.

```
:START
PAUSE Insert the disk to be copied in drive A
COPY A:*.* C:\LOTUS\JOHN
GOTO START
```

Here a continuous loop through the batch file occurs, requiring the user to press CTRL-BREAK to terminate the batch job and return to the DOS prompt level of operation. When we examine the IF command, we will use that command to construct a conditional branching example. This example will more fully demonstrate the capability of the GOTO command, which occurs when you use it with another batch command such as IF.

IF Command

The IF command is one of the most useful batch commands; it enables you to incorporate conditional branching into a batch file. The format of this command is:

```
IF [NOT] condition command
```

where the condition parameters are:

```
ERRORLEVEL number
String1 == String2
EXIST filespec
```

When the IF command is executed, the specified condition in the command is evaluated. If the condition is True, the command specified after the condition is executed. If the condition is False, the command specified after the condition is ignored and control is passed to the next line in the batch file.

The ERRORLEVEL number refers to the value of an optional exit code that a previously executed program may have set. When a program completes its execution correctly, it returns to DOS with an unset error flag. If the program terminates incorrectly, its error flag is set, causing DOS to examine a specific location to determine what the error code means.

Four DOS commands that return error codes if they terminate with an error that can be checked are FORMAT, REPLACE, BACKUP, and RESTORE. As an example of the use of error codes, assume you are creating a batch program file to automate BACKUP operations. In using an ERRORLEVEL X conditional operator in the IF command, the command will be executed if the returned error code is X or higher. Thus, the following simple batch file demonstrates the use of the ERRORLEVEL operator in the batch IF command.

```
BACKUP
IF ERRORLEVEL 1 ECHO!Backup Aborted!
```

The condition String1 == String2 in an IF command is true when both strings are identical. Unlike normal string comparisons, string values used in an IF command cannot include data redirection characters nor such command line delimiters as a comma, equal sign, colon, semicolon, and so on. As an example of the use of a string comparison condition in IF commands, consider the following batch file.

```
A>COPY CON: LISTDISK.BAT
IF %1 == LEFT GOTO DRIVEA
IF %1 == RIGHT GOTO DRIVEB
ECHO YOU OBVIOUSLY DO NOT KNOW WHAT TO DO
GOTO END
:DRIVEA
DIR A:>LPT1:
GOTO END
:DRIVEB
DIR B:>LPT1:
:END
```

If the computer's drives are stacked up and down, the program should be changed so

```
IF %1 == UP GOTO DRIVEA
IF %1 == DOWN GOTO DRIVEB
```

With this batch file, LISTDISK, the user simply enters the name of the file followed by the word LEFT or RIGHT to obtain a directory listing of an appropriately located drive. When this batch file is executed, the first entry in the command line following LISTDISK is assigned to the replaceable parameter %1. Thus, if the command line entry was

```
A>LISTDRIVE LEFT
```

the replaceable parameter %1 would be assigned the value LEFT. Because the first IF command is true, a branch to the label DRIVEA occurs in the program. The line after that label contains the DOS command DIR A:, with its output redirected to the printer printing the directory of that drive. Then the program branches to the end of the batch file. Similarly, if the user enters the command

```
LISTDISK RIGHT
```

the directory of the diskette in drive B will be printed. Although this was a trivial example, it illustrates the capability of using string comparisons in an IF command. In Section Two of this book we will develop several productivity techniques using string comparisons.

The last condition that can be tested in an IF command is the existence or nonexistence of a file specification. By using the conditions EXIST or NONEXIST, you can test whether a file is located in the default directory of a specified drive. The following portion of a batch file illustrates how you can test for the presence of a required file on a diskette in a specific drive and display an appropriate message if it is not found.

```
PAUSE Insert disk labeled XYZ in drive A
:START
IF EXISTS A:JAN88.WKS GOTO DOIT
PAUSE Wrong disk inserted in drive A
GOTO START
:DOIT
REM Program continues
```

SHIFT Command

The SHIFT command enables you to use more than 10 replaceable parameters in a batch file. When included in a batch file, this command changes the order in which the parameters %0 through %9 are replaced. That is, each SHIFT command causes every parameter to be shifted to the left. This may sound confusing, so let us examine the use of this command in a batch file. For example, suppose we create a batch file named SHOSHIFT.BAT, whose contents are

```
ECHO OFF
ECHO %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
SHIFT
ECHO %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
SHIFT
ECHO %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Next, let us assume you entered the following command line:

```
SHOSHIFT A B C D E F G H I J K
```

This would result in the display:

```
A>ECHO OFF
A B C D E F G H I
B C D E F G H I J
C D E F G H I J K
```

As indicated from the preceding example, you can use the SHIFT command to extend the number of replaceable parameters that can be used in a batch file. Although 10 replaceable parameters are sufficient for most applications, for creating complex batch processes the SHIFT command can be extremely useful.

CALL Command

Added with the introduction of DOS 3.3, this batch command permits one batch file to execute without requiring the first one to terminate. The format of this command is

```
CALL [d:][path]filename
```

The following portion of a batch file named RUN.BAT illustrates the use of the CALL command.

```
IF %1 == LOTUS CALL C:LOTUS
IF %1 == DBASE CALL D:DBASE
REM Program continues
```

When you enter the DOS command line `RUN LOTUS`, the file LOTUS on drive C will be called. Similarly, entering the DOS command line `CALL DBASE` invokes the batch file named DBASE located on drive D.

THE AUTOEXEC.BAT FILE

The AUTOEXEC.BAT file is a special batch file that automatically executes DOS commands as you boot up or do a warm boot. This special file is searched for by DOS each time your computer is powered on or a system reset is performed. If this file is located in the root directory of the drive from which DOS is initialized, it will be automatically executed.

The primary use of an AUTOEXEC.BAT file is to execute a series of DOS commands that are always initially required for the operation of your computer system. One example of a common use for an AUTOEXEC.BAT file is to display a menu system for novices to use who are not proficient with DOS commands. In Section Two we will examine several menu systems that can be generated through the use of an AUTOEXEC.BAT file.

SECTION TWO

Productivity Tips & Tricks

4

File and Subdirectory Operations

You saw in Chapter 3 that batch files can perform a variety of tasks related to files and subdirectories and can make the interface between the user and the computer much simpler. As an example, instead of issuing separate commands to first change the DOS current directory and then invoke a program, such as Lotus 1-2-3, you might simply create a batch file. Then, issuing one batch filename would automatically change the directory and execute the desired program.

Thus, a batch file by itself can increase your productivity by automating frequently performed functions. Once the file is stored, simply entering the name of the batch file can execute many instructions.

In this chapter we will examine the use of batch programs for operations normally performed on files and subdirectories. Some of the major topics covered in this chapter are development of several format protection schemes, a method to enhance disk-copying operations on systems that have only one disk drive, and methods to produce sorted directory listings, as well as several menu systems to simplify fixed-disk usage.

CAVEATS FOR BATCH FILES

Because batch files have user-defined names, it is important that your batch files do not have the same name as other DOS commands. Be careful also not to create two batch files with the same name to perform different functions. The remainder of this book will be designed so that each batch file created will have a different name and will not conflict with any DOS command.

Most of the batch files created here will be placed in their own subdirectory called BATUTIL. This subdirectory will be added to the search directory path when you use the DOS PATH command in the AUTOEXEC.BAT file, which is automatically executed when DOS is initialized. The PATH command tells DOS where to search for commands not located in the current directory. The PATH command we will use is

```
PATH C:\DOS;C:\BATUTIL
```

This PATH command tells the operating system to search two subdirectories for commands not in the current directory. The two subdirectories are \DOS, which is where users normally keep their operating system files, and \BATUTIL, which is where we will place all of the batch utilities being developed in this book. Note also that we are using a fixed disk system and our fixed disk drive is C:.

Be careful when you use the PATH command; each time you set it to a different path it forgets where it previously searched. Many application programs suggest that you load their software into their individual subdirectories, such as placing Lotus 1-2-3 in a subdirectory called \123, and then that you add the \123 subdirectory to the PATH command in the AUTOEXEC.BAT file. This philosophy is very good if you only have a few applications on your fixed disk, but very bad if you end up making a long PATH to be searched each time a command is issued. For this reason, our examples will generally show the invocation of applications by first using the DOS change directory (CD) command to make the application directory the current directory, then invoking the application program. This avoids the need for DOS to search a long PATH for the proper commands to execute.

FORMAT PROTECTION SCHEMES

Because formatting a diskette or hard disk destroys any previously recorded data, a mechanism for preventing inadvertent formatting is normally highly desirable. We will examine two format protection schemes that can be used to satisfy different operational requirements.

If you have a hard disk system, the FORMAT.EXE file from the DOS distribution diskette was probably copied to the root directory or to a subdirectory named UTILITY or DOS. If any person using the computer is at the C> prompt level and enters the command FORMAT, the resulting action will depend upon the version of DOS used. Although recent versions of DOS—from 3.0 onward—provide a warning, earlier versions of DOS simply initiate the format operation, destroying any data previously recorded on the hard disk. Even when a warning message is displayed, if it is late at night or if you are distracted as you perform this operation, it is quite possible that entering the wrong response to a warning message can destroy months or years of effort. As an example of this, consider the use of the FORMAT command under DOS Version 3.2. Here, entering the command FORMAT C: results in the following warning message:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE C: WILL BE LOST!
Proceed with Format (Y/N)?
```

If for some reason you were distracted and pressed the character Y, all data on drive C would be lost.

To prevent the inadvertent formatting of the hard disk you can first rename the FORMAT.EXE file to a less obtrusive name. Since the format process prepares a disk for use, you could rename it as you do any file:

```
RENAME FORMAT.EXE PREPARE.EXE
```

Next, you could create a one-line batch file that will restrict formatting to drive A. This is accomplished by creating a batch file using the name FORMAT for the file:

```
COPY CON: FORMAT.BAT
PREPARE A:%1 %2 %3 %4 %5 %6
```

In this example, entering the FORMAT command executes a batch file with that name. This file causes the FORMAT.EXE program that was previously renamed PREPARE.EXE to be executed. Up to six replaceable parameters will be passed to the renamed formatting program. This provides a way to pass six of the eight parameters that can be specified in the FORMAT command. In addition, the drive specifier A: will always be passed to the formatting program, restricting the use of the program to formatting diskettes in drive A.

Here is what you would see when you execute the previously created batch file; note the execution of the renamed FORMAT command:

```
C>format

C>prepare a:
Insert new diskette for drive A:
and strike ENTER when ready
```

Because the previously created one-line batch file executes the command PREPARE, it is displayed on the screen. If you feel this may cause confusion, you could prefix the PREPARE command with an ECHO OFF command, as shown here:

```
C>copy con: format.bat
echo off
prepare a:%1 %2 %3 %4 %5 %6
^Z
    1 File(s) copied
```

Here is how the modified PREPARE batch file would work:

```
C>format

C>echo off
Insert new diskette for drive A:
and strike ENTER when ready
```

Considering Device Names

Many users are familiar with device names, so they would probably enter the command FORMAT A: to format a diskette in drive A. If this entry occurred as the user was running the previously created batch file the error message `Invalid parameter` would be displayed and the batch program would terminate. This would occur because the drive specifier A: was included in the batch file. To permit additional flexibility in using the FORMAT command you can rename FORMAT.EXE as PREPARE.EXE and create the batch file listed in Program Listing 4.1 to execute the renamed FORMAT program.

Program Listing 4.1 **Considering Device Names**

```
ECHO OFF
IF %1==A: GOTO OK
IF %1==a: GOTO OK
IF %1==B: GOTO OK
IF %1==b: GOTO OK
ECHO This program only formats diskettes in drive A or B
GOTO END
:OK
PREPARE %1 %2 %3 %4 %5 %6
:END
```

Listing 4.1's batch file is more flexible for many users, because it enables users to format diskettes in drive A or drive B. In this batch file the IF statements test the replaceable parameter for the value A, a, B, or b and result in a branch to the label OK if a match occurs. If no match occurs the message `This program only formats diskettes in drive A or B` is displayed and a branch to the label END occurs.

ENHANCING SINGLE-DRIVE COPYING OPERATIONS

Although the PC XT and similar computers containing one floppy diskette drive are fine machines, they are not handy for making copies of diskettes. When you want to use the `DISKCOPY A: B:` command or its near equivalent, `COPY A:*.* B:`, DOS will use your one physical drive as two logical devices. Due to this, you will be treated to what is known as the "floppy shuffle" as you are alternately prompted to insert source and target diskettes as each file is copied. If there are many files on the diskette to be duplicated, you can conceivably spend 30 minutes or more performing the floppy shuffle.

To alleviate this inconvenience we will create a batch file that will automatically use a portion of the capacity of your hard disk as a second diskette drive. This batch file will create a temporary directory named TEMPCOPY under the root directory and copy all files from the diskette to be copied to that directory. Next, your batch file will prompt the user to place a formatted disk in drive A, copy the contents of the TEMPCOPY directory to the diskette in drive A, erase the contents of the temporary directory and then remove the directory.

Program Listing 4.2 shows the FASTCOPY.BAT batch program, created using the DOS line editor called EDLIN. This editor inserts line numbers as a reference to lines in a file; however, the actual file as saved does not contain line numbers. You can use EDLIN, copy data from the CON: device, or use a word processor capable of creating an ASCII file to create batch files.

Program Listing 4.2

Creating FASTCOPY.BAT

```
 1: ECHO OFF
 2: ECHO Place Diskette to be copied in Drive A
 3: PAUSE
 4: MD \TEMPCOPY
 5: COPY A:*.* C:\TEMPCOPY
 6: ECHO Place target Diskette in Drive A
 7: PAUSE
 8: ECHO Enter Y in response to next prompt
 9: ERASE C:\TEMPCOPY\*.*
10: RD C:\TEMPCOPY
```

Dos-not work on Tandy

In the batch file listed in Program Listing 4.2, line 1 turns the echoing of executed commands off and line 2 prompts the user to place the diskette to be copied into drive A. The PAUSE command in line 3 displays the prompt `Strike a key when ready . . . .` Once a key is pressed, line 4 causes the directory named TEMPCOPY to be created under the root directory.

The contents of the diskette in drive A are copied to the subdirectory TEMPCOPY by the COPY command in line 5. Next, the ECHO command in line 6 prompts the user to place the target diskette for the COPY operation in drive A. The PAUSE statement in line 7 again displays the prompt `Strike a key when ready . . . .`

After the files are copied to the diskette the batch file works on the assumption that they are of no use on the fixed disk and should be removed. Because the use of the ERASE command with the wildcard asterisk characters for filename and extension results in the generation of the DOS message `Are you sure (Y/N)?`, line 8 was included in our batch file to prompt the user how to respond to that message. Then, line 9 erases all files in the directory TEMPCOPY on drive C once the user enters the letter Y in response to the DOS message. Finally, line 10 removes the directory TEMPCOPY from the fixed disk.

The following sample session illustrates the execution of the FASTCOPY.BAT batch file. In this example, a diskette containing six files is copied. By using FASTCOPY instead of the DOS COPY command, you reduce the number of diskette insertions on a one-drive system from 12 to 2, which illustrates the major advantage in using this batch file.

```
C>FASTCOPY

C>ECHO OFF
Place DISKETTE to be copied in Drive A. . . .
Strike a key when ready . . .
A:AVG.PIC
A:PERF.WKS
A:WK1.PIC
A:WK2.PIC
A:WK3.PIC
A:WK4.PIC
        6 File(s) copied
Place Target Diskette in Drive A. . . .
Strike a key when ready . . .
C:\TEMPCOPY\AVG.PIC
C:\TEMPCOPY\PERF.WKS
C:\TEMPCOPY\WK1.PIC
C:\TEMPCOPY\WK2.PIC
C:\TEMPCOPY\WK3.PIC
```

```
C:\TEMPCOPY\WK4.PIC
      6 File(s) copied
Enter Y in response to next prompt
Are you sure (Y/N)?Y

C>
```

SUBDIRECTORY AND FILE SECURITY

To provide a very high measure of protection against the unauthorized reading of a file's contents, several vendors market programs that can encrypt data. In addition to costing as much as $100 per program, if you should forget or lose the encryption key you probably can never return the encrypted data to its original form. If you simply desire to prevent the inadvertent snooping of other users through one or more directories, you can use an undocumented feature associated with file and directory names to obtain a measure of protection.

By terminating the name of a directory or file with the character whose ASCII value is either 32 or 255, you increase the number of characters in the file or directory name by one. Here ASCII 32 is a blank character, and ASCII 255 is the NULL character. Both characters are nonprintable, so other users taking a directory listing can view the directory or filename. However, they will most likely fail to realize that an additional character is required to access the directory or file correctly.

Extending the Directory Name

To illustrate the addition of access protection to a directory, let us use the null character whose ASCII value is 255. To add the character whose ASCII value is 255 to a file or directory name you simply press and hold the ALT key and type the number 255 on the numeric keypad; however, for illustrative purposes the ASCII character value is displayed in parentheses. To see the result of this character addition to a directory name, you will create a subdirectory called MYDIR:

```
MD \MYDIRCHR$(255)
```

Note that in creating the subdirectory MYDIR, its name is actually six characters in length, because the null character CHR$(255) is not visible. Now assume that another user noticed the subdirectory MYDIR when he or she took a directory listing. If he or she entered the command:

```
CD \MYDIR
```

DOS would display the error message `Invalid Directory` because the user did not enter the final, invisible character whose ASCII value is 255 at the end of the directory name.

File Security

The same nonprinting character procedure used for directory names can be used to provide a measure of security to files. As an example, consider the creation of a batch file named R2D2 with the ASCII null character added to the end of the name of the batch file:

```
COPY CON: R2D2CHR$(255).BAT
ECHO OFF
REM YOU CAN'T LIST ME
^Z
```

Next, suppose another user attempts to use the TYPE command to obtain a listing of the contents of the file. If the user does not enter the correct filename to include the invisible fifth character in the name the message `File not found` will be displayed:

```
A>TYPE R2D2.BAT
File not found
A>
```

Thus, appending a nonprintable character to a directory name or filename can add a measure of privacy to your data. This in turn may enable you to avoid purchasing an encryption program when a simple measure of protecting data is all that is required.

DIRECTORY MAPPING

In this section we will develop a batch file to sort directories in several different ways, and we will devise a menu system to provide the user with several types of directory listings.

Sorting a Directory by Filename

The SORT command can be used to sort the filenames in the directory of our choice, but we must pass information about the filenames to the command. This can be accomplished by passing the DIR command to the SORT command using the piping feature built into DOS. (Chapter 1 described the concept of piping.) The following command sorts all files on the B drive in order by filename.

```
DIR B: ¦ SORT
```

Assuming our initial directory is as shown in Figure 4.1, after the directory is sorted, it would appear in alphabetical order, as indicated in Figure 4.2.

Because the default output from this command is to the screen display, we will see the sorted directory of drive B on the display. If we want to save this sorted directory, we could simply send the output of the SORT command to a file using the piping facilities of DOS:

```
DIR B: ¦ SORT >B:SORTDIR.TXT
```

This command would place the sorted directory listing of drive B into a file called SORTDIR.TXT. We could then display this file at any time with the TYPE command and get a sorted listing of all files on

Figure 4.1

Initial Directory Listing

```
A>dir b:

 Volume in drive B has no label
 Directory of  B:\

NOTEUSERS DIC       14   1-04-80   1:03p
JUNK               216   8-23-87   3:48p
TEMP      BAT       37  11-05-87   9:15p
FILE3     TXT      222  10-25-87   4:09p
GOODSTUF  DOC      263  10-25-87   4:21p
SAMPLE    XYZ       14   1-04-80   1:03p
         6 File(s)   723968 bytes free

A>
```

Figure 4.2 **Directory Sorted by Filename**

```
A>dir b: ¦sort


        6 Files(s)   723968 bytes free
 Directory of B:\
 Volume in drive B has no label
FILE3     TXT       222  10-25-87   4:09p
GOODSTUF  DOC       263  10-25-87   4:21p
JUNK                216   8-23-87   3:48p
NOTEUSRS  DIC        14   1-04-80   1:03p
SAMPLE    XYZ        14   1-04-80   1:03p
TEMP      BAT        37  11-05-87   9:15p

A>
```

drive B. As an alternative, we could include the line TYPE B:SORT DIR.TXT in a batch file to display the results of the sort automatically.

If we wanted to print the sorted directory, we could again use the SORT command with the piping facilities to send the output to the standard printer device PRN:

```
DIR B: ¦ SORT >PRN
```

This SORT command sends a sorted listing of the directory of drive B to the current standard printer, creating a screen dump of the alphabetically listed directory contents.

Sorting a Directory by Filename Extension

Suppose we want to sort the directory by filename extension to group all files of a type together. The SORT command can specify the starting column to sort on. Since the filename extension begins in column 9, the command to sort by filename extension would be:

```
DIR B: ¦ SORT /+9
```

Figure 4.3 shows a sample result of this command. Note that the period separating filename from extension does *not* appear in a directory entry.

Once again, it is simple to send this output to a file or to the printer using the DOS piping facilities:

```
DIR B: ¦ SORT /+9 >B:SORTDIR.TXT
or
DIR B: ¦ SORT /+9 >PRN
```

Here, the first command line directs the sorted output to the file SORTDIR.TXT on drive B; the second command line directs the sorted output to the printer. If a subdirectory is to be sorted, the command would be:

```
DIR B: \SUBDIR1 ¦ SORT /+9
or
DIR B: \SUBDIR1 ¦ SORT /+9 >B:SORTDIR.TXT
or
DIR B: \SUBDIR1 ¦ SORT /+9 >PRN
```

Figure 4.3

Directory Sorted by Filename Extension

```
A>dir b: ¦ sort /+9


JUNK              216   8-23-87   3:48p
TEMP     BAT       37  11-05-87   9:15p
NOTEUSRS DIC       14   1-04-80   1:03p
GOODSTUF DOC      263  10-25-87   4:21p
FILE3    TXT      222  10-25-87   4:09p
SAMPLE   XYZ       14   1-04-80   1:03p
        6 File(s)    723968 bytes free
 Volume in drive B has no label
 Directory of  B:\

A>
```

Sorting a Directory by File Size

To sort the directory by file size, the commands shown for sorting by filename extensions would be repeated, except for the column passed to the SORT command. To sort on the file size, we would inform the SORT command to begin in column 14:

```
DIR B: ¦ SORT /+14
```

This will result in a sorted directory listing of drive B on the screen, as shown in Figure 4.4.

Figure 4.4 **Directory Sorted by File Size**

```
A>dir b: ¦ sort /+14


NOTEUSRS DIC        14   1-04-80   1:03p
SAMPLE   XYZ        14   1-04-80   1:03p
TEMP     BAT        37  11-05-87   9:15p
JUNK               216   8-23-87   3:48p
FILE3    TXT       222  10-25-87   4:09p
GOODSTUF DOC       263  10-25-87   4:21p
 Directory of  B:\
        6 File(s)    723968 bytes free
 Volume in drive B has no label

A>
```

We could similarly send the output to a file or the printer with the following commands:

```
DIR B: ¦ SORT /+14 >B:SORTDIR.TXT
or
DIR B: ¦ SORT /+14 >PRN
```

A Batch Command for Different Directories and Outputs

You just saw how to sort directories in several different ways and how to send the output of the sorting to the screen, printer, or disk file. Now we will examine a batch command in which the user tells the SORT command which directory to sort and where the output should go (the screen or the printer). The batch file called SDIR.BAT can be created using the EDLIN facility or the COPY command. The contents of the file SDIR.BAT are:

```
ECHO OFF
CLS
DIR %1 ¦ SORT %2
```

Notice that the %1 and %2 parameters are values that the user passes to the SDIR.BAT batch file. To execute the file, type SDIR followed by the directory you wish to sort followed by the place you want the output sent. Some examples are:

```
SDIR B:\SUBDIR1 >PRN

SDIR B: >CON

SDIR B:\SUBDIR2 >B:\SUBDIR2\DIRSORT.TXT
```

The first example sorts the SUBDIR1 subdirectory on drive B and sends the output to the printer. The second example sorts all files on drive B and sends the output to the screen. The third example sorts the SUBDIR2 subdirectory of drive B and stores the output in a file called DIRSORT.TXT in the SUBDIR2 subdirectory. You could create a batch command similar to the preceding examples to sort by file size, as follows:

```
ECHO OFF
CLS
DIR %1 ¦ SORT /+14>%2
```

In this example after ECHO is turned off for batch file commands and the screen is cleared, a directory specified by using the first replaceable parameter is listed. Next, the directory is sorted by file size, and output is directed to the device specified by the second replaceable parameter.

A command to sort by filename extension would look like this:

```
ECHO OFF
CLS
DIR %1 ¦ SORT /+9>%2
```

A Sorting Menu System

You can easily develop a menu system that sorts in different ways when users select choices on a menu. Suppose you wanted a menu to look like Figure 4.5.

By using the menu illustrated in Figure 4.5 you could sort a directory by filename, size, or filename extension and have the output

Figure 4.5

Desired Menu System

```
DIRECTORY MAPPING FUNCTIONS
===========================

1 - Sort Directory by Filename to Screen
2 - Sort Directory by Filename to Printer
3 - Sort Directory by Size to Screen
4 - Sort Directory by Size to Printer
5 - Sort Directory by Filename Extension to Screen
6 - Sort Directory by Filename Extension to Printer
7 - Display Wide Directory Listing
0 - END - Return to DOS
```

sent to the screen or the printer, or you could obtain a wide listing of your directory. For example, by simply typing a 1 followed by the name of the directory you wish to sort you would obtain a sorted listing on the screen. A batch file can display a menu on the screen and ask the user to select an option. Program Listing 4.3 illustrates the creation of a batch file using the EDLIN text editor.

Program Listing 4.3 **Batch File to Display Menu**

```
1:*echo off
2: cls
3: type sdir.txt
4: prompt ENTER NUMBER FOLLOWED BY DIRECTORY $q$g
```

The first line in the file turns off the echo of commands to the screen, so users do not see the commands being executed. Line 2 clears the display. Line 3 is used to put the menu on the screen. In this example the menu was placed in a file called SDIR.TXT (Program Listing 4.4) and the TYPE command in DOS is used in line 3 of Program Listing 4.3 to display it on the screen. Line 4 changes the standard DOS prompt from A: to words that tell the user what to do next (for example, enter the number of your choice from the menu and follow the number with the directory you wish to have sorted).

Program Listing 4.4 **Directory Mapping Functions**

```
1:*
2:
3:          DIRECTORY MAPPING FUNCTIONS
4:          ============================
5:
6:          1 - Sort Directory by Filename to screen
7:          2 - Sort Directory by Filename to printer
8:          3 - Sort Directory by Size to screen
```

```
 9:       4 - Sort Directory by Size to printer
10:       5 - Sort Directory by Filename Extension to screen
11:       6 - Sort Directory by Filename Extension to printer
12:       7 - Display Wide Directory listing
13:       0 - END - Return to DOS
```

A batch file named 1.BAT will be created next to perform the sorting; it will look as shown in Program Listing 4.5.

Program Listing 4.5

1.BAT File to Perform Sorting by Filename to Screen

```
1:*echo off
2: cls
3: dir %1 ¦sort ¦more
4: pause
5: cls
6: type sdir.txt
```

The first line in the 1.BAT file turns off the echo of the commands to the screen so that the user doesn't see what commands are issued by the batch file. The second line clears the screen and the third line actually sorts the specified directory. Notice that line 3 looks very similar to the sorting batch files developed earlier, but also contains a MORE command. When a large directory is being sorted and more than 24 files are to be listed, the list will scroll off the top of the screen as the successive lines are displayed. MORE is a DOS command that stops the screen from scrolling when it is full and tells the user that `-MORE-` information is to be displayed after looking at this screen. To continue the display and scrolling, simply press any key and the next full screen will be shown.

The fourth line contains a PAUSE command. This batch command causes a temporary halt to the processing of batch commands until the user is ready to continue. When the PAUSE command is executed, it will tell the user to `Strike any key when ready. . .` on the screen. The PAUSE is needed at this point in the batch file because we want to look at the sorted directory display just produced. Line 5 is a clear screen command; if the PAUSE command were not placed just prior to the CLS command you would not have time to look at your sorted directory before the screen was cleared. The sixth and final command in the 1.BAT file again displays the menu of choices to the user.

The peculiar name of this batch file—1—was selected to correspond to an option on the main menu. That is, if the user responds to the main menu by selecting option 1, the batch file with that name executes.

Similarly, we will create additional batch files with numeric names to correspond to the other options on the main menu. The file named 2.BAT sorts a directory and sends the results to the printer. The 2.BAT file is shown in Program Listing 4.6. Notice that there is no need to include a clear screen command (CLS) at the beginning of this procedure because the output is going directly to the printer. There is also no need to PAUSE after the sort is printed because the user is not using the screen to look at the sorted directory listing. The command on line 2 performs the sort of the directory specified by the user and sends the output to the printer. Line 3 clears the screen and line four redisplays the menu for another selection.

Program Listing 4.6 **2.BAT File to Perform Sorting by Filename to Printer**

```
1:*echo off
2: dir %1 ¦sort >prn
3: cls
4: type sdir.txt
```

Program Listings 4.7 through 4.10 show the contents of files 3.BAT through 6.BAT, respectively. These files are very similar to files 1.BAT and 2.BAT, except for the directory that is sorted. Program Listing 4.11 shows the contents of the file 7.BAT, where line two results in the specified directory being displayed on the screen in a wide format.

Program Listing 4.7

3.BAT File to Sort Directory by Size to Screen

```
1:*echo off
2: cls
3: dir %1 ¦sort /+14 ¦more
4: pause
5: cls
6: type sdir.txt
```

Program Listing 4.8

4.BAT File to Sort Directory by Size to Printer

```
1:*echo off
2: dir %1 ¦sort /+14 >prn
3: cls
4: type sdir.txt
```

Program Listing 4.9

5.BAT File to Sort Directory by Filename Extension to Screen

```
1:*echo off
2: cls
3: dir %1 ¦sort /+9 ¦more
4: pause
5: cls
6: type sdir.txt
```

Program Listing 4.10

6.BAT File to Sort Directory by Filename Extension to Printer

```
1:*echo off
2: dir %1 ¦sort /+9 >prn
```

```
3: cls
4: type sdir.txt
```

Program Listing 4.11 **7.BAT File to Display Wide Directory Listing**

```
1:*echo off
2: dir %1 /w
3: pause
4: cls
5: type sdir.txt
```

Program Listing 4.12 shows the contents of file 0.BAT, which probably requires a little explanation. Since choice 0 on your menu is `End--Return to DOS` you want to remove your menu from the screen and put the standard DOS prompt back when this option is selected. Line 2 in the file performs the clear screen operation, whereas line 3 resets the DOS prompt back to the default drive letter (A in our case, but C if you have a hard disk drive), followed by a greater than symbol (>). At this point you have exited your sorted directory menu system and returned control to DOS.

Program Listing 4.12 **File 0.BAT**

```
1:*echo off
2: cls
3: prompt $n$g
```

THE FILE LOCATOR

Since the introduction of hierarchical directory structures with DOS Version 2.0, finding files that are not in the current directory has become more complex and time-consuming. Fairly expensive commercial file-finding software is available, but there is a less costly solution. Through the use of a batch file that takes advantage of the results of executing the CHKDSK command, you can develop a mechanism to locate files easily in a complex hierarchical directory structure.

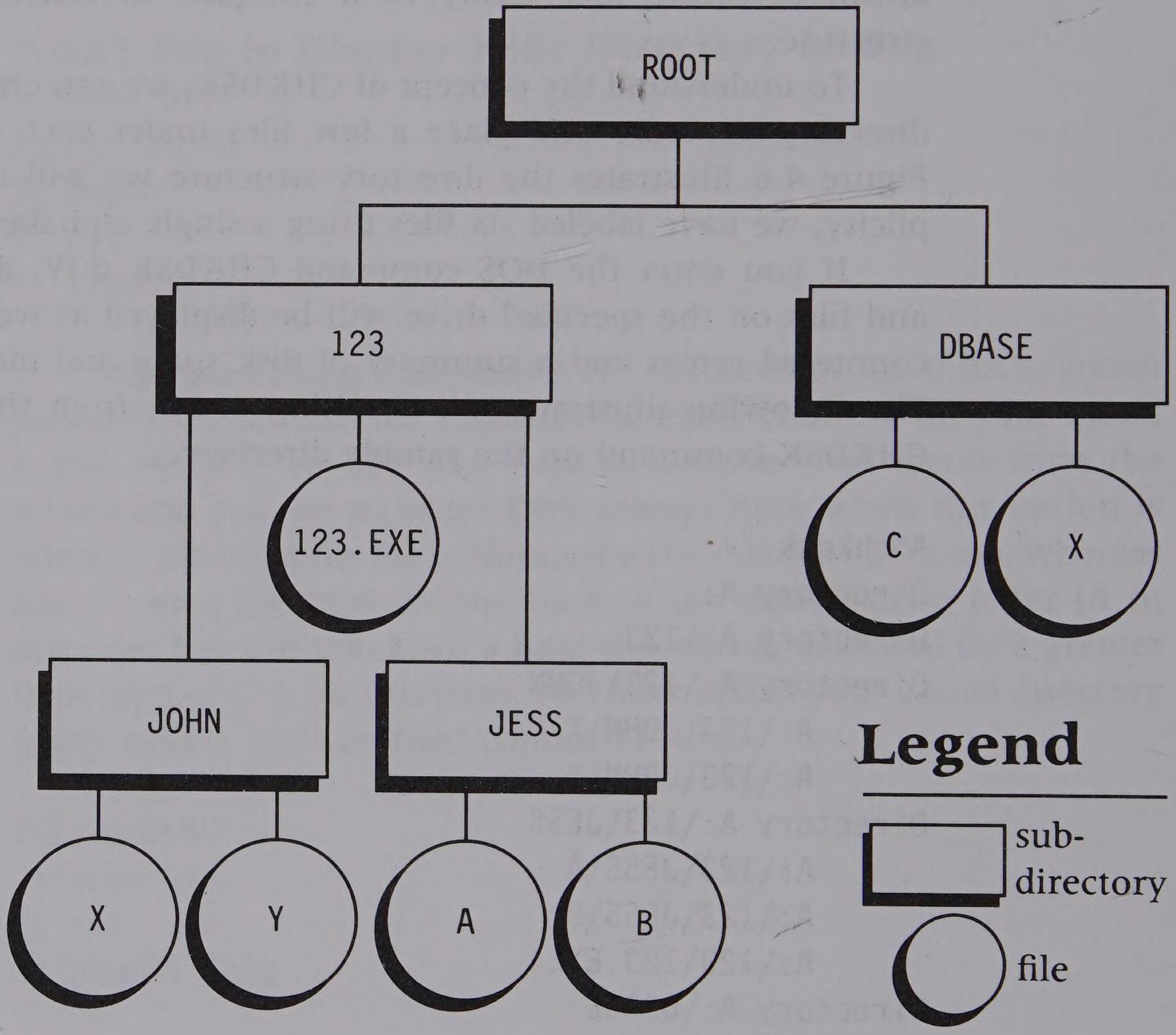
To understand the concept of CHKDSK, we can create a sample directory structure and place a few files under each subdirectory. Figure 4.6 illustrates the directory structure we will use. For simplicity, we have labeled six files using a single alphabetic character.

If you enter the DOS command CHKDSK d:/V, all directories and files on the specified drive will be displayed as well as any encountered errors and a summary of disk space and memory usage. The following illustrates the resulting action from the use of the CHKDSK command on the sample directory.

```
A>chkdsk /v
Directory A:
Directory A:\123
Directory A:\123\JOHN
      A:\123\JOHN\X
      A:\123\JOHN\Y
Directory A:\123\JESS
      A:\123\JESS\A
      A:\123\JESS\B
      A:\123\123.EXE
Directory A:\DBASE
      A:\DBASE\C
      A:\DBASE\X

   730112 bytes total disk space
     4096 bytes in 4 directories
     7168 bytes in 7 user files
   718848 bytes available on disk
```

Figure 4.6 Sample Directory Structure


ROOT
123
DBASE
123.EXE
C
X
JOHN
JESS
X
Y
A
B
Legend
sub-directory
file

```
524992 bytes total memory
477520 bytes free
```

When CHKDSK executes, it displays a list of all directories and files contained on the disk. By first directing the output of the CHKDSK command to a temporary file, you can process that file with the FIND command. Program Listing 4.13 illustrates the creation of the batch file named LOCATE.BAT, which is constructed to provide a mechanism to rapidly locate any file on a disk.

Program Listing 4.13

Creating a File Locator Program

```
A>copy con: LOCATE.BAT
echo off
if "%1" == "" goto :error
echo FILE LOCATOR COMMENCING...PLEASE WAIT...
chkdsk /V >temp.$$$
find "%1" temp.$$$
erase temp.$$$
echo END OF SEARCH...
goto :end
:error
echo FILENAME MUST BE ENTERED AFTER LOCATE...TRY AGAIN...
:end
^Z
        1 File(s) copied

A>
```

In Program Listing 4.13 the DOS COPY command copies the input from the keyboard (CON:) to the file named LOCATE.BAT. The first line turns the echo of commands from the batch file off. The second line checks whether the user entered a filename to search for. If no filename is entered (the %1 parameter is null) the batch LOCATE program transfers to an error routine (:error) that tells the user to enter the required filename. The third line echoes a message to the user that the LOCATE program is starting to run. The fourth line actually executes the CHKDSK command and sends the output to a file called TEMP.$$$. Line 5 uses the DOS FIND command to find the filename entered by the user in the TEMP.$$$ file. This FIND command will show the user every line in the TEMP.$$$ file that contains the filename being searched for. The sixth line erases the TEMP.$$$ file, which is no longer needed. The seventh line tells the user that the search for the filename is completed. Line 8 bypasses the error message display previously described and goes to the end of the program.

The use of the LOCATE.BAT file is illustrated in the following.

```
A>locate X


A>echo off
FILE LOCATOR COMMENCING...PLEASE WAIT...


---------- temp.$$$
        A:\123\JOHN\X
        A:\123\123.EXE
        A:\DBASE\X
END OF SEARCH...


A>
```

Note that entering the command LOCATE X located two files named X—one under the subdirectory JOHN and another in the subdirectory DBASE. Also notice that the filename 123.EXE was also shown as being found by our LOCATE command. This is because it contains the X character in its name and was picked up by the FIND command in line five of the LOCATE.BAT file.

Expanding the Locate Capability

You can expand the previously created batch file to locate either directories or files by name. To do so let us first define a command format that will be used as follows, with the items in braces indicating a choice of one of the two.

$$\text{LOCATE}\left\{\begin{matrix}\text{DIR}\\ \text{FILE}\end{matrix}\right\}\text{NAME}$$

The preceding command format requires two replaceable parameters—one for DIR or FILE, the other for the name of the directory or file to locate. Program Listing 4.14 illustrates the revised LOCATE.BAT file.

Program Listing 4.14

Revised LOCATE.BAT File

```
*echo off
if "%2" == "" goto :error
if "%1" == "DIR" goto :dir
if "%1" == "FILE" goto :file
echo "DIR" OR "FILE" MUST BE ENTERED AFTER LOCATE...TRY AGAIN
goto :end
:dir
chkdsk /v ¦FIND "Directory" >temp.$$$
find "%2" temp.$$$
erase temp.$$$
goto :end
:file
chkdsk /v >temp.$$$
find "%2" temp.$$$
erase temp.$$$
goto :end
:error
```

```
18: echo "NAME OF DIR" OR "FILENAME" TO LOCATE MISSING...TRY AGAIN
19: end
```

Line 1 of this revised file is similar to the previously created file, because it uses the ECHO command to turn off the echo of batch commands to the screen. Line 2 checks to see if the user entered a filename or directory name to search for. If no name was entered, an error message is issued and the LOCATE command is ended. Lines 3 and 4 check to see if the correct syntax of the LOCATE command was used and the word DIR or FILE was entered immediately after entering LOCATE. If DIR was entered, control passes to the part of the file that processes the directory searching following the label :dir. If FILE was entered, control passes to the part of the file that processes filename searching following the label :file.

Lines 5 and 6 tell the user that the DIR or FILE keywords must be entered after LOCATE and the program is ended by a branch to the label :end. Line 8 uses the CHKDSK and FIND commands to find all directories on the disk and place them in a file called TEMP.$$$. Line 9 searches the TEMP.$$$ file for the directory name the user is looking for and displays it on the screen. Lines 10 and 11 clean up previous work by deleting the TEMP.$$$ file and ending the program. Lines 13 through 15 are the same as in the previous LOCATE.BAT file (with the exception that the parameter is %2 here) and are used to find the filename you are looking for.

Here are several examples of executing the revised LOCATE.BAT file.

```
A> locate

A>echo off
"NAME OF DIR" OR "FILENAME" TO LOCATE MISSING...TRY AGAIN

A>locate DIR
```

```
A>echo off
"NAME OF DIR" OR "FILENAME" TO LOCATE MISSING...TRY AGAIN

A>locate 123

A>echo off
"DIR" OR "FILE" MUST BE ENTERED AFTER LOCATE...TRY AGAIN

A>

A>locate FILE

A>echo off
"NAME OF DIR" OR "FILENAME" TO LOCATE MISSING...TRY AGAIN

A>locate

A>echo off

---------- temp.$$$
Directory A:\123
Directory A:\123\JOHN
Directory A:\123\JESS

A>locate FILE X

A>echo off

---------- temp.$$$
      A:\123\JOHN\X
```

The File Locator

```
        A:\123\123.EXE
        A:\DBASE\X

A>
```

CREATING FILE REFERENCE COMMANDS

Although DOS provides commands for direct execution of the most widely used operations, it can't cover all situations. You may, therefore, find it convenient to modify the action of an existing command or to create a new command to meet your needs. In this section, we will create two new DOS commands that will illustrate how you can use the power of batch files to modify or tailor DOS to your specific requirements.

Creating a Display Command

An easy way for a user to see what is in a file is to display its contents on the screen. Unfortunately, DOS does not have a DISPLAY command, but uses the TYPE command to display files. We can create a batch file with the name DISPLAY.BAT, so we will create our own DISPLAY command that uses the DOS TYPE command.

When you issue the TYPE command to display a file on the screen, it first looks for the file to be displayed. If the file cannot be found it issues the message:

```
File not found
```

In our DISPLAY command we might want to give the user a little better description of the error, so we might create our DISPLAY command as illustrated in Program Listing 4.15.

Program Listing 4.15

Creating a Display Command

```
1:*echo off
2: if not exist %1 goto :ERROR
3: type %1
4: goto :END
5: :ERROR
6: echo File %1 Does Not Exist...Display Failed...
7: :END
```

Line 1 simply turns off the echo of batch commands to the screen so that the user does not see them. Line 2 checks to see if the file the user wishes to display actually exists. If the file does not exist, control is passed to an error routine, at line 5, which tells the user using the echo command on line 6 that the file does not exist and that the display has failed. When the file does exist, line 3 executes the TYPE command to display the file on the screen. An example of the execution of this DISPLAY command is:

```
A>display pay87.dat

A>echo off
File pay87.dat Does Not Exist...Display Failed...

A>display b:pay87.dat

A>echo off
MELVIN    FRED     X2015551212A06
JONES     JONATHAN  3055451412B10
NACHES    PAUL      4158201495A06
PORTER    MARYANNE Q3055451416B10
HOOVER    HERBERT  H2015551218A06

A>
```

Although the DISPLAY command's function is dependent upon the TYPE command, its error message is now much more explicit than that provided by DOS. In addition, by renaming the command to DISPLAY it more accurately reflects the operation of the command and will probably be easier to remember by novice users.

The MOVE Command

Although DOS contains a COPY command, it lacks a MOVE command which would physically relocate a file. In comparison, the COPY command makes a duplicate of a file on another directory or another disk without affecting the original file.

Because moving a file is equivalent to first copying a file and then deleting the original file, you can use the DOS COPY and ERASE commands to construct a MOVE command. If you wish to add a MOVE command to DOS, you can create a batch file named MOVE.BAT as illustrated in Program Listing 4.16.

Program Listing 4.16

Creating a MOVE Command

```
1:*echo off
2: copy %1 %2
3: if not exist %2 goto notok
4: del %1
5: goto ok
6: :notok
7: echo MOVE OPERATION FAILED...
8: :ok
```

In this example, the replaceable parameters %1 and %2 will be used for the source and destination file specifications. After the COPY operation is performed, the IF command in line 3 is used to determine the presence or absence of the destination file. If the file does not exist, a branch to the label NOTOK in line 6 occurs, resulting in the display of the error message `MOVE OPERATION FAILED`. If the target file exists, no branch occurs as a result of the IF command, causing the program to execute the DEL command in line 4 to erase the source file. Then a branch to the label OK occurs, in effect terminating the program.

As an example of the use of the MOVE.BAT file, assume you entered the command

```
MOVE A:PAY87.DAT B:PAY87.DAT
```

In this example the replaceable parameters %1 and %2 are assigned the values A:PAY87.DAT and B:PAY87.DAT, respectively. This causes the file PAY87.DAT to be copied from drive A to drive B. Next, when the IF command in the batch file verifies the existence of the file on drive B, the DEL command erases the file from drive A.

The example in Figure 4.7 first shows a directory listing of drive B and a directory of all files ending in .DAT on drive A. Next, the MOVE command is executed to move the PAY87.DAT file from drive A to drive B. Finally, a directory of drive B shows the PAY87.DAT file, and a directory of all files ending in .DAT on the A: drive shows that the PAY87.DAT file has been removed from A.

MASTER MENUS FOR FIXED DISK SYSTEMS

The use of fixed disk (also called hard disk) systems has become much more prevalent recently because of the decreasing cost and increased availability of fixed disk storage devices. Many users today are putting multiple applications on their fixed disks, resulting in a requirement for an easy way to navigate between applications. You can design a simple batch menu system to assist in this task, or you can create a

Figure 4.7 **Sample Actions of the MOVE Command**

```
A>dir b:

 Volume in drive B has no label
 Directory of  B:\

PRTLABEL BAK      242  11-18-87   9:29p
PRTLABEL BAT      228  11-18-87   9:50p
123          <DIR>     11-08-87  11:59a
DBASE        <DIR>     11-08-87  11:59a
        4 File(s)    716800 bytes free

A>dir a:*.dat

 Volume in drive A is TREBLIG
 Directory of  A:\

PAY87    DAT      206  11-08-87   2:53p
        1 File(s)    291840 bytes free


A>move a:pay87.dat b:pay87.dat

A>echo off
        1 File(s) copied
```

Figure 4.7 (*continued*)

```
A>dir b:

 Volume in drive B has no label
 Directory of  B:\

PAY87    DAT       206  11-08-87   2:53p
PRTLABEL BAK       242  11-18-87   9:29p
PRTLABEL BAT       228  11-18-87   9:50p
123          <DIR>      11-08-87  11:59a
DBASE        <DIR>      11-08-87  11:59a
        5 File(s)    715776 bytes free

A>


A>dir a:*.dat

 Volume in drive A is TREBLIG
 Directory of  A:\

File not found
```

more complex system that accomplishes a similar function in a more elegant manner. In this section we will examine several methods of creating master menus, beginning with a simple method that helps you navigate among directories.

Suppose you have a PC with a large fixed disk and wish to use

Master Menus for Fixed Disk Systems

Lotus 1-2-3, dBASE III, and MultiMate. A simple menu scheme would offer you ways to choose between each of these applications by simply selecting a letter or number from the menu. An example of this menu is shown in Figure 4.8.

Once the menu in Figure 4.8 is displayed, the user should be able to simply enter the letter of his or her choice and press ENTER to use a selected application. To create the menu we will create files named A.BAT, B.BAT, and C.BAT to begin each of the applications and a batch file called X.BAT to enable the user to exit the MENU system and return to the DOS prompt. A file called MENU.TXT will be used to display the menu on the screen and allow the user a choice. The MENU.BAT file is shown in Program Listing 4.17.

Figure 4.8 Sample Menu for Selecting Applications

```
                    GILBERT'S MAIN MENU
==============================================================

                     A - Lotus 1-2-3

                     B - dBASE III

                     C - MultiMate


                     X - Exit MENU System to DOS


 Enter your selection -
```

Program Listing 4.17

MENU.BAT File

```
1:*echo off
2: cls
3: type menu.txt
4: prompt Enter your selection -
```

Line 1 of MENU.BAT turns off the echo of commands to the screen so the user does not see what commands are issued. Line 2 clears the screen and line 3 places the menu on the screen by typing a file called MENU.TXT. This file is given in Program Listing 4.18.

Program Listing 4.18

The Text File MENU.TXT Displayed by MENU.BAT

```
 1:*                  GILBERT'S MAIN MENU
 2:=============================================================
 3:
 4:                     A - Lotus 1-2-3
 5:
 6:                     B - dBASE III
 7:
 8:                     C - MultiMate
 9:
10:
11:
12:                     X - Exit MENU System to DOS
```

Line 4 of the MENU.BAT file changes the DOS prompt from the standard of default disk drive and greater than symbol to the words `Enter your selection -`. When the menu is placed upon the screen, the user is asked to select one of the choices on the menu, as shown in Figure 4.9.

The X.BAT file is used to exit from the menu and return control to the DOS system. In reality, however, control is still at the DOS level but the prompt has been changed. All the X.BAT file has to do

Figure 4.9

Display Resulting from Execution of MENU.BAT

```
                    GILBERT'S MAIN MENU
=============================================================

                       A - Lotus 1-2-3

                       B - dBASE III

                       C - MultiMate


                       X - Exit MENU System to DOS


 Enter your selection -
```

is return the prompt to its standard symbol—the current drive letter and the greater than symbol (>). The X.BAT file is shown in Program Listing 4.19. Line 1 simply turns the echo of commands to the screen off. Line 2 clears the screen and line 3 changes the DOS prompt back to its standard symbol, the default disk drive followed by the greater than symbol.

Program Listing 4.19

X.BAT File

```
1:*echo off
2: cls
3: prompt $n$g
```

The A.BAT file invokes the Lotus 1-2-3 program. This batch file is illustrated in Program Listing 4.20. Line 1 turns the echo of commands to the screen off, whereas line 2 clears the screen. Then, line 3 uses the DOS CD command to change the current directory to the subdirectory where the Lotus 1-2-3 software is located. In this example, it is assumed that Lotus 1-2-3 resides in the directory named 123. Next, line 4 executes the Lotus software. When the Lotus 1-2-3 system is exited, DOS executes the next command in the batch file, resulting in line 5 changing the current directory back to the highest level or root directory. Finally, line 6 issues the MENU command that executes the MENU.BAT file and places the master menu back onto the screen.

Program Listing 4.20

A.BAT File

```
1:*echo off
2: cls
3: cd \123
4: lotus
5: cd \
6: menu
```

The B.BAT file in Program Listing 4.21 invokes the dBASE III program. This batch is structured exactly the same as the A.BAT file, except that line 3 changes the current directory to the DBASE subdirectory, whereas line 4 executes the dBASE software. Lines 5 and 6 are used to return to the master menu.

Program Listing 4.21

B.BAT File

```
1:*echo off
2: cls
3: cd \dbase
4: dbase
5: cd \
6: menu
```

The last file required to create the simple menu is the C.BAT file. This file is used to invoke MultiMate word processing software and is illustrated in Program Listing 4.22. Note that it is the same as the A.BAT and B.BAT files, except that line 3 changes the current directory to the MULTIMATE subdirectory and line 4 executes the MultiMate software.

Program Listing 4.22

C.BAT File

```
1:*echo off
2: cls
3: cd \mm
4: mm
5: cd \
6: menu
```

Using BASIC

A different method of developing a menu system uses the BASIC interpreter that comes with most versions of DOS. You start the interpreter by issuing the command BASIC, or BASICA for the advanced BASIC, or GW-BASIC for the BASIC provided with many versions of MS-DOS for PC clones.

Program Listing 4.23 shows an example of a BASIC program for controlling software selections on Jane's PC. This program has three options that can be selected: word processing, spreadsheet or database functions. The BASIC SHELL command is used to execute a batch program corresponding to the selection made from the menu. When you terminate the word processor, spreadsheet, or database function, control is returned to the BASIC program and the menu is again displayed on the screen.

Program Listing 4.23 BASIC Language Program for Controlling Software Selection

```
10 REM **************************************************************
20 REM *
30 REM * Main Menu for JANE's PC
40 REM *
50 REM **************************************************************
60 CLS
70 PRINT "                         JANE's PC MAIN MENU"
80 PRINT "-----------------------------------------------------------------"
90 PRINT
100 PRINT
110 PRINT "                1 - WORD PROCESSING"
120 PRINT
130 PRINT "                2 - SPREADSHEET"
140 PRINT
150 PRINT "                3 - DATABASE"
160 PRINT
170 PRINT
180 PRINT
190 PRINT
200 INPUT "Enter the NUMBER of your choice... ";N
210 IF N=1 THEN SHELL "C:\BATUTIL\WP.BAT"
220 IF N=2 THEN SHELL "C:\BATUTIL\SS.BAT"
230 IF N=3 THEN SHELL "C:\BATUTIL\DB.BAT"
240 GOTO 60
250 END
```

Lines 10 through 50 are simply remarks within the program using the BASIC REM statement. These lines give anyone who looks at the program information that the author of the program feels is important. In this example, it's simply a remark that this is the main menu for Jane's PC. Other information could be added such as the name of the program and the date it was created, as well as the name of the author or programmer.

Line 60 is the BASIC CLS statement, which simply clears the screen on the PC similar to the DOS CLS command. Lines 70 through 190 use the PRINT command to place the menu on the PC screen.

Line 200 uses the INPUT command to obtain a response from the user. The INPUT command first asks the user to enter a number of his or her choice from the menu and waits for the user to enter a number into the variable N. Line 210 checks to see if the number entered is a 1 and if so executes the SHELL command to run a batch file named WP.BAT under the BATUTIL subdirectory on the C: drive. Lines 220 and 230 work in a manner similar to line 210. That is, they check if the number entered is a 2 or 3 and execute batch files named SS.BAT and DB.BAT, respectively, upon encountering the predefined number. If the user enters anything other than a 1, 2, or 3, the GOTO command at line 240 is executed. This GOTO causes the program to start over at line 60, which first clears the screen and then places the menu back onto the screen. Line 250 is an END command that indicates the end of the program.

Although the DOS ECHO command performs the same general function as the BASIC PRINT statement, using BASIC provides you with greater flexibility in performing different tasks. As an example, BASIC programming statements enable you to compare keyboard input against predefined values and to then perform branching operations based upon the value of the input. In comparison, batch file processing only permits data input in the form of replaceable parameters that are entered when the batch file is executed. Thus, you can easily construct a BASIC program to accept keyboard input to perform different operations, whereas attempting to do the same when using batch files can be either extremely difficult or impossible.

A second advantage in the use of BASIC for creating menus is the ease of screen control from that language. Through the BASIC COLOR statement you can easily set foreground, background, and border colors, as well as perform highlight and blinking operations. While these operations can also be performed from within batch files and are covered in Chapter 5, many readers familiar with BASIC will prefer to perform those operations from within that language. Thus, the use of the SHELL command is the key to obtaining the ability of increased menu programming flexibility from within BASIC.

The SHELL Command

The SHELL command allows a command or batch file to be run from within a BASIC program and is the key to how the menu works properly. In essence, the BASIC program is simply "put to sleep" while the batch file or DOS command is executed. When the batch file is finished with its work, or when the DOS command is completed, the DOS command EXIT is used to return to the BASIC program, "wake" it up, and allow it to continue operating at the point where it was last running.

Figure 4.10 shows what the BASIC program looks like on the screen when it is executed.

Suppose the program is saved on your hard disk under the name MENU.BAS. To have the program run we simply enter the command

```
BASIC MENU
```

This starts the BASIC interpreter and automatically loads and executes the program MENU.BAS. Of course, the BASIC interpreter must be stored in your current directory or pointed to by the DOS PATH command; the program MENU.BAS must also be in your current directory or in a directory pointed to by the DOS PATH command.

Instead of entering the command ourselves, we could set up an AUTOEXEC.BAT file:

```
1: echo off
2: path = \DOS
3: BASIC MENU
```

Figure 4.10 Resulting BASIC Program Display

```
                          JANE's PC MAIN MENU
--------------------------------------------------------------------


               1 - WORD PROCESSING

               2 - SPREADSHEET

               3 - DATABASE


Enter the NUMBER of your choice... ?
```

This file turns off echoes of commands to the screen in line 1. In line 2 the DOS PATH command is used to set up search paths to the subdirectory \DOS, which is where the BASIC interpreter is assumed to reside. Then, line 3 executes the BASIC interpreter and the MENU.BAS program.

Program Listing 4.24 illustrates the contents of the WP.BAT file. Line 1 turns off echoes of commands to the screen, line 2 uses the DOS change directory command to set the \MM subdirectory as our current directory, and line 3 executes the word processor program. Line 4 is the DOS EXIT command, which is executed when the word processor is finished.

Program Listing 4.24 **WP.BAT File**

```
1:*echo off
2: cd \mm
3: mm
4: exit
```

This EXIT command returns control to the BASIC program that called the batch file and is necessary because the SHELL command was used within the program. Similarly, the SS.BAT and DB.BAT files are shown in Program Listings 4.25 and 4.26. These files operate exactly like the WP.BAT file, but they use different subdirectories and call a spreadsheet and database, respectively. That is, the SS file changes the directory to 123 while the DB file changes the directory to DBASE. Then, each file issues the appropriate command to initiate the program in each subdirectory.

Program Listing 4.25

SS.BAT File

```
*echo off
cd \123
lotus
exit
```

Program Listing 4.26

DB.BAT File

```
*echo off
cd \dbase
dbase
exit
```

Increasing Menu Functionality

To add additional functionality to the BASIC menu we will increase the documentation with REM statements and add an error routine to catch simple errors. Program Listing 4.27 shows an updated BASIC program that retains the name MENU.BAS.

Program Listing 4.27

Revised BASIC Program

```
10 REM ****************************************************************
20 REM *
```

```
30 REM * Program name:   MENU.BAS
40 REM * Author:         Tom Domore
50 REM * Last update:    December 12, 1987
60 REM *
70 REM * This program is the main menu for JANE's PC
80 REM *
90 REM *****************************************************************
100 CLS
110 PRINT "                           JANE's PC MAIN MENU"
120 PRINT "------------------------------------------------------------------
130 PRINT
140 PRINT
150 PRINT "             1 - WORD PROCESSING"
160 PRINT
170 PRINT "             2 - SPREADSHEET"
180 PRINT
190 PRINT "             3 - DATABASE"
200 PRINT
210 PRINT
220 PRINT
230 PRINT "             0 - EXIT...Return to DOS"
240 PRINT
250 PRINT
260 INPUT "Enter the NUMBER of your choice... ";N
270 IF N=0 THEN CLS :SYSTEM
280 IF N=1 THEN SHELL "C:\BATUTIL\WP.BAT"  :GOTO 100
290 IF N=2 THEN SHELL "C:\BATUTIL\SS.BAT"  :GOTO 100
300 IF N=3 THEN SHELL "C:\BATUTIL\DB.BAT"  :GOTO 100
310 PRINT
320 PRINT "*** ERROR ***  -->  You must enter a NUMBER between 0 and 3..."
330 PRINT
340 INPUT "Press <ENTER> or <RETURN> key to try again...";X$
350 GOTO 100
360 END
```

Lines 10 through 90 are all remarks, but this time we have added information that may be beneficial in a business environment. The program name is called MENU.BAS, the author is Tom Domore, and the date the program was last changed was December 12, 1987. This information could be used by someone who later was given the task of changing the menu on all PCs in the company to add additional menu options for such applications as communications and graphics.

Line 100 still clears the screen, while lines 110 through 250 use the PRINT command to put the menu on the screen. Notice that the option "0 - EXIT . . . Return to DOS" was added to the menu. You may have a user who wishes to work at the DOS level and is knowledgeable about PCs and this option now offers an easy and "clean" way to safely exit the BASIC MENU program. Line 260 uses the INPUT command to request that the user make a selection from the menu and places the value selected into the variable N. Line 270 is used to check for the selection of 0, indicating that the user wishes to exit to DOS. If the choice is 0, the screen is cleared with the CLS command and the SYSTEM command returns control to the DOS system.

SYSTEM is actually a BASIC command that says to close the BASIC program, stop the BASIC interpreter, and turn control back over to the disk operating system. Lines 280 through 300 each check for a selection from the menu and, based upon the number selected, execute a batch file using the SHELL command. When the batch file is completed, the next command after the SHELL command is executed and, in this case, it is the GOTO 100 command. This returns the program to the clear screen routine and redisplays the menu so the user can make another selection.

This simple modification to the program now allows any selection that is not a number 0 through 3 to fall through the program to line 310. Lines 310 through 340 print an error message on the screen telling the user that he must enter a number between 0 and 3. The INPUT command at line 340 simply allows the program to pause long enough for the user to read the error message before line 350 is executed. Line 350 returns control to line 100, which clears the screen and places the menu back on the screen. An example of the execution of this program is shown in Figure 4.11.

Master Menus for Fixed Disk Systems

Figure 4.11

Execution of Revised BASIC PROGRAM

```
                         JANE's PC MAIN MENU
----------------------------------------------------------------------

                  1 - WORD PROCESSING

                  2 - SPREADSHEET

                  3 - DATABASE


                  0 - EXIT...Return to DOS


Enter the NUMBER of your choice... ? 7
*** ERROR ***  -->  You must enter a NUMBER between 0 and 3...

Press <ENTER> or <RETURN> key to try again...?
```

There is still one problem with this program as it is written. Suppose someone enters a letter or special character and not a number as planned for. In this situation, BASIC is looking for a numeric value (or number) for the variable N in line 260 and any character other than a numeric value triggers a BASIC error that displays a question mark on the screen and prompts the user to `Redo from start`, meaning to enter again. This message is not very meaningful to someone who is not familiar with computers and the BASIC programming language. Figure 4.12 illustrates the program's response to a user entering `QUIT` instead of `0` for EXIT. Note that the `?Redo from start` message generated by the BASIC interpreter is followed by the display of the `Enter the NUMBER of your choice . . .?` mes-

Figure 4.12 **BASIC Language Generated Error Message**

```
                              JANE's PC MAIN MENU
--------------------------------------------------------------------


              1 - WORD PROCESSING

              2 - SPREADSHEET

              3 - DATABASE


              0 - EXIT...Return to DOS


Enter the NUMBER of your choice... ? QUIT
?Redo from start
Enter the NUMBER of your choice... ?
```

sage a second time. The novice user might well ask, is this a request to start all over or should I simply enter a number?

To eliminate the potential confusion, avoid the possibility of generating the error message. To do so, change line 260 of the program to accept any character, whether numeric or alphabetic, by using a string variable N$ in the statement. Then, if a string variable is used lines 270 through 300 of the program must be modified to check for the numbers 0 through 3 in the form of string values. The changes to lines 260 through 300 of the program previously listed in Program Listing 4.27 are

```
260 INPUT "Enter the NUMBER of your choice. . . ";N$
270 IF N$="0" THEN CLS :SYSTEM
```

```
280 IF N$="1" THEN SHELL "C:\BATUTIL\WP.BAT" :GOTO 100
290 IF N$="2" THEN SHELL "C:\BATUTIL\SS.BAT" :GOTO 100
300 IF N$="3" THEN SHELL "C:\BATUTIL\DB.BAT" :GOTO 100
```

As a result of the modification of the program to accept keyboard input as strings, let us again execute the program and enter invalid data. Figure 4.13 illustrates the execution of the BASIC program designed to accept string data. Note that the improper keyboard entry of QUIT does not result in the generation of the BASIC `?Redo from start` error message. Since N$ does not equal the string 1, 2, or 3, the error message at line 320 in the program is executed. Then, once

Figure 4.13 **Execution of BASIC Program That Accepts String Data**

```
                           JANE's PC MAIN MENU
-----------------------------------------------------------------

                  1 - WORD PROCESSING

                  2 - SPREADSHEET

                  3 - DATABASE

                  0 - EXIT...Return to DOS


Enter the NUMBER of your choice... ? QUIT

*** ERROR ***  -->  You must enter a NUMBER between 0 and 3...

Press <ENTER> or <RETURN> key to try again...?
```

the user presses the ENTER key the program will branch to line 100, clear the screen, and redisplay the menu.

Strictly BATCH

Now let us create a menu of strictly batch files and place this menu and all associated batch files in a subdirectory called \BATUTIL. Once this is accomplished we could execute this menu with the CD command to the \BATUTIL subdirectory and enter the command MAINMENU to run the MAINMENU.BAT file. We could also create a batch file called MENU.BAT that simply runs the MAINMENU.BAT file by including the one line command MAINMENU in that file. Another method to get the menu to appear automatically on the screen is to create an AUTOEXEC.BAT file in the root directory that automatically changes to the \BATUTIL directory and executes the MAINMENU.BAT file. An example of an AUTOEXEC.BAT file that performs these operations whenever the computer is powered on or the system is reset appears in Program Listing 4.28.

Program Listing 4.28 **AUTOEXEC.BAT File**

```
1: echo off
2: cls
3: path = \DOS;\BATUTIL
4: cd \batutil
5: MAINMENU
```

The AUTOEXEC.BAT file turns off the echo of commands to the screen in line 1. Line 2 clears the screen and line 3 sets the DOS search paths for commands and files to first look at the \DOS subdirectory and then look at the \BATUTIL subdirectory. Line 4 changes the current directory to the \BATUTIL subdirectory, and line 5 executes the MAINMENU.BAT file to place the menu on the screen.

Program Listing 4.29 shows the contents of the MAINMENU.BAT file. Lines 1 and 2 turn echo of commands off and clear the screen, respectively. Line 3 types out a file called MAINMENU.TXT with the text that forms the menu. Line 4 changes the DOS prompt from the standard A> (C> on a hard disk) to the words `Enter your selection please . . .`. Now, whenever the menu is displayed the user will see the `Enter your selection please . . .` prompt instead of the C> prompt. Using the DOS PROMPT command to change the prompt has made the menu much more meaningful and easy for a novice to use. An experienced user will quickly recognize that the computer is still at DOS level and only the prompt has been changed. Any DOS command can still be given at this prompt.

Program Listing 4.29

MAINMENU.BAT File

```
1:*echo off
2: cls
3: type mainmenu.txt
4: prompt Enter your selection please . . .
```

Figure 4.14 shows the contents of the MAINMENU.TXT file, which in this example contains a menu for Tom. Notice that six options with numbers are available for use, and an option for U for Utilities or 0 to exit to DOS can be chosen. Each of the options is similar to our previous example of batch menus in which entering a number invokes batch files called 1.BAT, 2.BAT, 3.BAT, and so on.

Program Listing 4.30 shows the 1.BAT file. Here, lines 5, 6, and 7 are used to return to the main menu after the selected spreadsheet software is exited. Similarly, files 2.BAT through 6.BAT would be constructed, the only differences between them being in the location where the change of directory occurs (line 3) and the command used to invoke the application program (line 4). But what about this option 0, which supposedly allows us to return to DOS? Program Listing

Figure 4.14

Text File MAINMENU.TXT

```
*
                      MAIN MENU FOR TOM
====================================================================


   1 - SPREADSHEET                   U - Utilities

   2 - WORD PROCESSING

   3 - DATABASE

   4 - PROFS

   5 - HARVARD GRAPHICS

   6 - PAGEMAKER                     0 - Return to DOS


```

4.31 shows the 0.BAT file. Lines 1 and 2 simply turn off echo of the following commands to the screen and clear the screen. Line 3 tells the user that the system is returning to DOS control, while line 4 changes the current directory to the root directory. Line 5 again uses the DOS PROMPT command to change the prompt back to the current disk drive and the greater than symbol (C> in this case). Remember that we changed the prompt in MAINMENU.BAT and this simply changes the prompt back to its original value.

Program Listing 4.30

1.BAT File

```
*echo off
cls
cd \123
```

Master Menus for Fixed Disk Systems

```
4: lotus
5: cls
6: cd \batutil
7: type mainmenu.txt
```

Program Listing 4.31

0.BAT File

```
1:*echo off
2: cls
3: echo ....Returning to DOS....
4: cd \
5: prompt $n$g
```

Program Listing 4.32 shows the U.BAT file, which brings up a Utility menu. Lines 1 and 2 simply turn off the echo of commands and clear the screen, while line 3 displays the UTIL.TXT file that contains the utility menu. The prompt has not been changed since MAINMENU.BAT, and the user is still asked to `Enter your selection please . . ..`

Program Listing 4.32

U.BAT File

```
1:*echo off
2: cls
3: type util.txt
```

Figure 4.15 shows the contents of the UTIL.TXT file used to create a utility menu. Note that each of the utilities in this menu uses letters for its execution. This is because these batch files are M.BAT, D.BAT, S.BAT, and so on. They are not explained here, but using the concepts provided in this book you could easily generate your own utility batch files for this menu. Then, to return to the MAINMENU.BAT file you would create a two-line R.BAT file

Figure 4.15

UTIL.TXT File

```
*
                        UTILITY MENU
  ===================================================================


     M - Move files

     D - Display file

     S - Sort Directory

     P - Printer Control

     L - Label generation

     F - File locator                R - Return to MAIN MENU


```

whose contents would be ECHO OFF followed by MAINMENU on the second line.

DIRECTORY NAVIGATION

Many times when using DOS you may wish to execute a program or access a particular file that is in a different directory than the one currently in use. Knowing what directories are on a disk and their proper spelling can help tremendously when the time comes to change to a different directory using the DOS CD command. Navigation of directories can be made much easier with a batch file that lists all directories on a disk. This only works, though, if the file permits you to see the directory names when making the new directory current.

Program Listing 4.33 shows a batch file called NAVIGATE.BAT, which can be used to list all directories on a disk.

Program Listing 4.33

NAVIGATE.BAT File

```
1:*echo off
2: cls
3: echo All directories on %1 disk are:
4: echo -------------------------------
5: tree %1 ¦find "\"
6: echo -------------------------------
7: echo Use CD \XXXX\XXXX to make any directory current...
```

Line 1 of the batch file turns the echo of commands to the screen off; line 2 uses the DOS CLS command to clear the screen. Lines 3 and 4 simply put a heading at the top of the screen telling the user that what follows is a listing of all directories on the disk. The %1 is a parameter that is passed to the NAVIGATE.BAT file when it is executed. It is the disk drive of the disk whose directories are to be listed. Line 5 uses the DOS TREE command and points to the %1 disk drive and sends the output from the TREE command to the FIND filter. The filter finds and displays on the screen only output from the TREE command that contains the character \. This character is the separator for directory and subdirectory names and is always part of a directory name. Line 6 simply draws a line at the end of the directory listing, and line 7 tells the user to use the DOS CD command to change to the directory of his or her choice. To execute the NAVIGATE.BAT file for a listing of all directories on drive B, the user would enter

```
NAVIGATE B:
```

The execution of the NAVIGATE batch file is shown in this sample output:

```
All directories on B: disk are:
-------------------------------
Path: B:\123
Path: B:\123\JOHN
Path: B:\123\JESS
Path: B:\DBASE
-------------------------------
Use CD \XXXX\XXXX to make any directory current...

A>
```

A hard disk usually contains a lot of directories and subdirectories, and a display of their names may more than fill the screen. This causes some names to "roll off" the top. A simple modification of the batch file to correct this problem is shown in Program Listing 4.34. Notice in line 5 that the MORE filter has been added after the TREE and FIND commands. This command causes the screen to stop when it is full and tell the user that more output is on the next screen. When the screen pauses the user has the chance to see what directory names are available before they "roll off" the screen.

Program Listing 4.34

Revised NAVIGATE.BAT File

```
1:*echo off
2: cls
3: echo All Directories on %1 disk are:
4: echo -------------------------------
5: tree %1 ¦find "\" ¦more
6: echo -------------------------------
7: echo Use CD \XXXX\XXXX to make any directory current...
```

5

Screen Control

Although this chapter will present only one program (and an enhanced version) for screen control, the program can be integrated with others covered in this book and illustrates the use of several key concepts. In particular, the program illustrates the method by which you can use ANSI extended screen and keyboard control codes to control the foreground, background, and certain video attributes on a color display. Although screen control can also be accomplished through programs developed with a high-level language, such as BASIC or C, it is often more practical and easier to accomplish this task with a batch file. You can integrate this batch file as a module into another batch file that first sets the colors of a display and then performs another predefined sequence of operations. Alternatively, a master batch file can use a one-line command to invoke the batch file that controls the screen attributes.

The section of this chapter entitled Examining Screen Control Codes deviates slightly from our prior format by illustrating concepts instead of presenting an actual program. The reason is that the color display section illustrates the method required for generating ANSI control codes. The subsequent section instead provides details about the use of other ANSI control codes. You can use the program in this chapter as an example for constructing other programs that use the control codes.

SCREEN AND KEYBOARD CONTROL

The key to invoking screen and keyboard control sequences through DOS is use of the American National Standards Institute (ANSI) system file contained on your operating system diskette. This file, ANSI.SYS, provides a translation facility between a set of standardized codes for console operations and the internal codes recognized by the design of the IBM PC series to perform such operations. The functions include cursor positioning, device status reporting, text erasing, keyboard key reassignment, and screen control. In effect, ANSI.SYS enables you to develop programs that use the same sequence of characters to perform a given operation on many different computer systems, provided that they all have this translation facility.

To use these standardized screen and keyboard control codes, you must install the device driver ANSI.SYS, which performs the translation. You install ANSI.SYS by placing the following statement in the configuration file on your computer:

```
DEVICE = [d:][path]ANSI.SYS
```

where d: represents the drive on which the file is located, and the path is the route to the ANSI.SYS file if it is located somewhere other than the root directory.

ANSI Control Sequence Format

Under the ANSI standard, screen- and keystroke-control sequences take the format:

```
ESC [parameters COMMAND
```

In this command format, ESC is the standard abbreviation for the escape character, whose ASCII code is hexadecimal 1B. If you convert hexadecimal 1B to binary you obtain

```
      1            B          Hexadecimal
     / \          / \
  0 0 0 1      1 1 0 1        Binary
```

Next, if you convert the resulting binary value to decimal, you obtain the value 27. Thus, this conversion explains why a high-level program directs the value CHR$(27) to the printer to start an escape code sequence that places the printer in a particular mode of operation.

Under DOS, you can use the PROMPT command to send an escape code to the display, because PROMPT $e is equivalent to the ESC character. Therefore, by directing output to the printer through the DOS redirection facility, you can use PROMPT $e to send an escape character to the printer from within the operating system.

Returning to the ANSI control sequence format, the character that follows the escape code is the left bracket ([), which indicates the start of the parameters. The parameters are numeric integers unless you do not specify a parameter value or you specify a value of 0. In either case, a default value is used for the parameter. Finally, the COMMAND is an alphabetic string that identifies the command to the ANSI.SYS device driver.

Although Controlling a Color Display is limited to illustrating how the attributes on a color display can be changed, read on in this chapter for additional information concerning cursor control and screen erasure under ANSI.SYS. By applying the concepts you will learn about color displays to the other screen control codes, you can develop batch files to perform cursor control and other screen-related operations that are normally considered possible only with a high-level language program.

CONTROLLING A COLOR DISPLAY

When the IBM PC was first introduced, a large majority of systems sold used a monochrome display. For that reason, many application programs did not provide the ability to control the foreground and background colors on a color display, so that running these programs on a PC color system resulted in the default display of white characters on a black background. Although many modern application programs now include a setup feature so you can select foreground and background colors, a large number of programs do not provide this capability. Thus it is helpful to obtain some external mechanism to control a color display; such a mechanism is the subject of this section.

The PROMPT Command

The key to setting the color on a display is the PROMPT command. Although the use of this command is primarily associated with displaying a new DOS prompt, you can also use the command to generate an escape character sequence. If you place the line

```
DEVICE = ANSI.SYS
```

in your CONFIG.SYS file, you can use the ANSI extended screen and keyboard control codes that, in addition to providing a standard method of cursor control, also include a mechanism for setting the graphics rendition of the display. If you use the DOS PROMPT command to generate the required escape character sequences, you can use the Set Graphics Rendition option of the ANSI.SYS driver to set foreground and background colors on a color monitor. In addition, if you desire, you can expand upon the program presented here to include the setting of such attributes as blink and reverse video, or you can make data invisible by setting foreground and background colors equal to one another. This would permit, as an example, an invisible field for password entries as a security feature of your program.

The control sequence required by the ANSI.SYS device driver is of the form

```
ESC [#;...;#m
```

Table 5.1

Character Attributes of ANSI.SYS Set Graphics Rendition

Parameter	Meaning
0	All attributes off (normal white on black)
1	Bold on (high intensity)
4	Underscore on (IBM Monochrome Display only)
5	Blink on
7	Reverse video on
8	Canceled on (invisible)
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

The parameters and their meaning in the Set Graphics Rendition mode are listed in Table 5.1.

As an example of the use of the PROMPT command to set a screen color, assume you want to set the foreground color to red. The ANSI.SYS required escape code is

```
ESC [31m
```

If you use the DOS prompt command, PROMPT $e is equivalent to the ESC character, so that PROMPT $e[31m is functionally the same as the ANSI.SYS escape sequence. Using this equivalent capability, let us construct the color display program COLOR.BAT, Program Listing 5.1.

Program Listing 5.1

COLOR.BAT Program

```
goto b%1
:NORMAL
prompt $e[0m
goto :END
:FBLACK
prompt $e[30m
goto :END
:FRED
prompt $e[31m
goto :END
:FGREEN
prompt $e[32m
goto :END
:FYELLOW
prompt $e[33m
goto :END
:FBLUE
```

```
prompt $e[34m
goto :END
:FMAGENTA
prompt $e[35m
goto :END
:FCYAN
prompt $e[36m
goto :END
:FWHITE
prompt $e[37m
goto :END
:BBLACK
prompt $e[40m
goto :END
:BRED
prompt $e[41m
goto :END
:BGREEN
prompt $e[42m
goto :END
:BYELLOW
prompt $e[43m
goto :END
:BBLUE
prompt $e[44m
goto :END
:BMAGENTA
prompt $e[45m
goto :END
:BCYAN
prompt $e[46m
goto :END
```

```
:BWHITE
prompt $e[47m
:END
shift
if not "."==".%1" goto :%1
prompt
```

By entering the command COLOR followed by the color name prefixed by an F or B, you can set the appropriate foreground or background color. As an example, entering the command COLOR FWHITE sets the foreground color of the display to WHITE.

Using the batch file you can include more than one parameter in the command line. Thus, entering the command COLOR FWHITE BBLUE would result in a white foreground on a blue background.

As constructed, the COLOR.BAT file only works with standard-brightness colors. To extend the number of colors supported, you can add the label :BRIGHT followed by the DOS prompt PROMPT $e[1m and GOTO:END branch to the file. Program Listing 5.2 illustrates this modification to the program. This modification enables you to select high-intensity foreground and background colors. As an example, entering the command COLOR BRIGHT FRED would first turn on high intensity and then select RED for the foreground color. To change from high to low intensity you can enter the command COLOR NORMAL followed by the name of the color to change to low intensity. If COLOR NORMAL is entered by itself, the display is changed to its default foreground and background colors.

Program Listing 5.2

Modified COLOR.BAT Program

```
goto %1
:BRIGHT
prompt $e[1m
goto :END
```

```
:NORMAL
prompt $e[0m
goto :END
:FBLACK
prompt $e[30m
goto :END
:FRED
prompt $e[31m
goto :END
:FGREEN
prompt $e[32m
goto :END
:FYELLOW
prompt $e[33m
goto :END
:FBLUE
prompt $e[34m
goto :END
:FMAGENTA
prompt $e[35m
goto :END
:FCYAN
prompt $e[36m
goto :END
:FWHITE
prompt $e[37m
goto :END
:BBLACK
prompt $e[40m
goto :END
:BRED
prompt $e[41m
```

```
goto :END
:BGREEN
prompt $e[42m
goto :END
:BYELLOW
prompt $e[43m
goto :END
:BBLUE
prompt $e[44m
goto :END
:BMAGENTA
prompt $e[45m
goto :END
:BCYAN
prompt $e[46m
goto :END
:BWHITE
prompt $e[47m
:END
shift
if not "."==".%1" goto :%1
prompt
```

EXAMINING SCREEN CONTROL CODES

As previously discussed, you can use predefined control codes and the ANSI.SYS device driver to perform cursor control operations as well as erase text from the screen through DOS. Here we will examine the control sequences necessary to perform such operations.

Cursor Positioning

Table 5.2 lists the cursor positioning command formats that can be used under ANSI.SYS to control cursor positioning through DOS.

You can use the DOS prompt command to generate cursor positioning codes, in much the same way as shown in our program example for generating screen attribute codes. As an example, the DOS command

```
PROMPT $e [5C
```

moves the cursor forward five columns without changing the row position. Similarly, the DOS command

```
PROMPT $e [10,20f
```

Table 5.2

Cursor Positioning Command Formats

Function	Command Format	Description
Move to Position	ESC [#;#H	Moves cursor to position specified by the parameters
Cursor Up	ESC [#A	Moves cursor up specified number of positions
Cursor Down	ESC [#B	Moves cursor down specified number of positions
Cursor Forward	ESC [#C	Moves cursor forward specified number of positions within row
Horizontal and Vertical	ESC [#;#F	Moves cursor to the position specified by the parameters

positions the cursor to row 10, column 20. Note that the pound signs contained in Table 5.2 are replaced by the integer numeric values desired for the actual cursor movement.

Text Erasure

Two codes in ANSI.SYS can be used to erase text from the screen. Table 5.3 contains the control sequences and code formats required to erase text from the screen.

The ability to move the cursor and erase text from the screen through DOS may appear to be a trivial capability at first glance. In actuality, there are instances in which this capability may prove to be very practical. Consider a program that provides a screen display, of which you only want to print a portion. Using the cursor positioning and erasure display capability within DOS you can selectively edit the screen prior to obtaining a printout of the modified display.

SUMMARY

The ANSI.SYS device driver file on your DOS diskette controls screen and keyboard functions, such as cursor positions and text erasure.

Table 5.3

Text Erasure Command Formats

Function	Command Format	Description
Clear Screen	`ESC [2J`	Erase screen and reposition cursor at home location
Erase in Line	`ESC [K`	Erase from cursor to end of line

The DOS PROMPT command sends an escape code to the screen; PROMPT $e sends an escape character to the printer from within DOS.

PROMPT also can set the graphics rendition and colors of your display by generating escape character sequences; it also can set attributes such as blink and reverse video.

The COLOR.BAT program listed in this chapter sets foreground and background colors. With some modifications, COLOR.BAT can also set high-intensity foreground and background colors.

6

Printer Control

In this chapter we'll examine a variety of techniques that help you maximize the use of the printers attached to your personal computer.

The first section explores a method of swapping printer ports. The program developed in this section is most applicable for those users who have both a letter-quality and a dot matrix printer attached to their computer system, and who use application programs that have no built-in ability to select appropriate printer ports.

The second section examines the construction of a batch file that can invoke the printer spooler program provided on the DOS diskette.

The remainder of this chapter illustrates a method of creating mailing labels and shows a mechanism for easy control of the different printing modes built into most printers. Like all programming examples presented in this book, the printer productivity techniques presented in this chapter can be combined with other user programs or other productivity modules covered in this book to develop routines necessary to satisfy your specific requirements.

PRINTER PORT SWAPPING

Suppose you have two printers connected to your computer—a letter-quality printer and a dot matrix draft printer. Assume you need to use each at different times, but they are connected to different parallel ports of your PC. In this example, we will use a batch file to call a BASIC program that can examine available printer ports and swap the printers if you so desire. To accomplish the swap, we will use the DOS COPY command to create a batch file named PRINTER.BAT; it automatically executes a BASICA program named PRTSWAP.BAS.

```
A>COPY CON: PRINTER.BAT
```

The only entry in this batch file is the command to call the BASIC program:

```
BASICA PRTSWAP.BAS
```

This command calls BASICA and executes a program saved in the current directory as PRTSWAP.BAS. The BASIC program begins, in lines 100 and 110 in Program Listing 6.1, by defining two messages to describe which printer is currently available.

Program Listing 6.1

BASIC Program to Swap Printer Ports

```
100 LET LQMSG$="Current printer is LETTER QUALITY Printer"
110 LET DMMSG$="Current printer is DOT MATRIX Printer"
120 DEF SEG=%H40
130 LET A=PEEK(8)
140 LET B=PEEK(9)
150 LET C=PEEK(10)
160 LET D=PEEK(11)
170 CLS
180 PRINT "SWAP BETWEEN LETTER-QUALITY AND DOT MATRIX PRINTERS"
190 PRINT "---------------------------------------------------"
200 PRINT
210 IF A=120 AND B=2 THEN PRINT LQMSG$
220 IF A=188 AND B=3 THEN PRINT DMMSG$
230 PRINT
240 PRINT "would you like to change printers? (Y/N)";
250 INPUT X$
260 IF X$="N" OR X$="n" THEN GO TO 310
270 POKE 8,C
280 POKE 9,D
290 POKE 10,A
300 POKE 11,B
310 SYSTEM
320 END
```

Lines 120 through 160 define the base address segment for BASIC to use and PEEK commands to store the values in locations 8, 9, 10, and 11 of the memory address segment. These locations are where the pointers to the parallel ports controlling the two printers are located. The PEEK command is used to look at the actual values in certain locations of memory in the computer and the LET command assigns those values to the variables A, B, C, and D.

Lines 170 through 200 simply clear the screen and tell us what the program is supposed to accomplish. In programming terms, this is known as placing a heading on the screen.

Lines 210 and 220 check the values in the memory areas that were "peeked" at earlier and saved in the variables A, B, C, and D; these values are compared to the values relevant to your letter-quality or dot matrix printer. Line 240 then prints a message telling you which printer is currently active. Note that your system may use different values; after you install two different printers, you should load BASIC and use the PEEK commands as direct statements to look at the values in these memory locations. Use the values returned in A, B, C, and D in the program.

Lines 230 through 260 ask you if you would like to change printers and if your answer is (N)o, the program terminates.

If your answer is (Y)es, lines 270 through 300 swap the printers by reversing the values previously in the memory locations that were "peeked" at.

Line 310 is the end of the program and returns control to DOS and the batch commands that started this program.

To execute the program, simply enter the command `PRINTER` at the DOS prompt.

PRINTER SPOOLING

The term *spool* comes from an old procedure used on mainframes called Simultaneous Peripheral Operation on Line. The purpose of the procedure is to prevent a slow peripheral, such as a printer or modem, from hogging all of the computer's resources until the peripheral's task is completed. Nowadays, *spooling* refers to any of several methods of transferring data from a file to a printer in such a way that you can run other programs while the printer is running.

Instead of sending output from an application program directly to the printer, character by character, at the rate the printer can accept it—50 characters per second (cps) for daisy-wheel printers, 120 to 400 cps for dot matrix printers—you send as much output as possible to a memory buffer at high speed (many thousands of characters per second). The printer takes characters at its own speed from the buffer, leaving the computer free to perform other tasks until it needs to refill the print buffer with another block of output.

There are two main classes of print spoolers: one is an external unit that has its own microprocessor and memory. This microprocessor runs a program to get blocks of text from the host computer and feed them to one or more printers. The other type of spooler uses part of the host computer's own memory as a print buffer, and software that sends characters to the printer during times when no other task of higher priority is running (for example, while the computer is waiting for you to enter characters from the keyboard). Some printers have internal buffers that can hold 8,000, 16,000, or even 32,000 characters; such printers allow you to reduce, or even eliminate, the use of the computer's main memory for print buffers. In this chapter we'll be mainly concerned with spoolers that use part of your computer's memory for buffering.

DOS PRINT Command

In recent releases of DOS the designers have recognized the need for a print spooler of some type; they have added the PRINT command to the operating system. The PRINT command can print a file on a line printer attached to the computer. It can also create a holding area to queue up multiple files for printing and can select which device is to be used for printing if more than one printer is available on a computer.

When you issue the PRINT command after DOS is started, you are prompted to verify what the name of the device is and where the printed output is to go. The DOS default for this is PRN, and you can accept it by pressing the ENTER or RETURN key. Other devices could be the console screen (CON), a different line printer (LPT2), or even a communication port (COM1). Once you select the desired output device, the PRINT command installs the PRINT routines in a resident part of memory and tells whether a file is in the queue to be printed:

```
C>print
Name of list device [PRN]:
Resident part of PRINT installed
PRINT queue is empty
```

To print more than one file at a time on the printer, you can simply issue the PRINT command with the desired filenames after it. Following is a PRINT command line that includes five different filenames after it. The files are placed in the print queue in the order specified by the command. DOS also lists all files in the queue for reference purposes.

```
C>print menu.bat 1.bat 2.bat autoexec.bat config.sys
    C:\MENU.BAT is currently being printed
    C:\1.BAT is in queue
    C:\2.BAT is in queue
    C:\AUTOEXEC.BAT is in queue
    C:\CONFIG.SYS is in queue
```

If you were using the PRINT spooler and wanted to know what files were in the queue for printing, you could simply issue the PRINT command without any parameters, as in the following example. The result is a listing of all files in the queue, in the order in which they will be printed. The PRINT command provides additional parameters that you can use to add files to or delete files from the print queue.

```
C:\MENU.BAT is currently being printed
C:\1.BAT is in queue
C:\2.BAT is in queue
C:\AUTOEXEC.BAT is in queue
C:\CONFIG.SYS is in queue
```

As a further example of the power of PRINT, suppose each week you use a word processor to change your budget and then print the budget with the most recent changes. You could use the PRINT command in a batch file to send automatically a particular file to the printer immediately after it is updated. Suppose you want to do work with your word processor while the budget is being printed. An example of a batch file to accomplish this task is shown in Program Listing 6.2. This batch file is called BUD.BAT.

Program Listing 6.2

BUD.BAT File

```
*cd \mm
mm
cd \
print <device.prn budget.prn
cd \mm
mm
cd \
```

Line 1 in the file uses the CD command to switch to the word processing directory, while line 2 executes the word processor program—in this example, mm—that is the actual name of the executable program to invoke the word processor. Line 3 returns control to the root directory so the PRINT command can be used (this is not necessary if you have previously set up a PATH command to point to where the DOS PRINT command is located). Line 4 issues the PRINT command and specifies the file to be printed. Lines 5 and 6 change back to your word processing directory and again execute the word processor program, thereby allowing you to work while the printing is ongoing. Finally, line 7 in the batch file returns you to the root directory when the word processing work is completed.

Although most of the entries in the batch file are self-explanatory, line 4 requires some additional elaboration. The PRINT command is there, along with the file to be printed (BUDGET.PRN), but what is the `<DEVICE.PRN` in the middle of this command? Well, remember that the PRINT command must be told which device you want to print to (usually PRN), and it always asks you the question the first time you use PRINT. You don't have to answer the question, because the < symbol tells DOS to accept its next input (the answer to the question `Name of list device`) from a file called DEVICE.PRN. This file has been placed in the root directory (that is why the batch file went back to the root directory) and looks like the following example. Notice that this file contains only one line and in it is the value `PRN:`.

```
C>edlin device.prn
End of input file
*1
        1:*PRN:
*
```

The PRINT command takes the `PRN:` from the file DEVICE.PRN to answer the `Name of list device` question and begins printing the BUDGET.PRN file. The batch file does not stop for the user to answer any questions and continues to restart the word processor. An example of the execution of part of this batch file is

```
C>cd \

C>print <device.prn budget.prn

    C:\BUDGET.PRN is currently being printed

C>cd \mm
```

One of the key advantages of using the DOS PRINT command is the ability to obtain a multitasking capability. That is, you can select one or more files for printing and continue to perform other computer operations while the files print. By tailoring the example presented in this section to your particular application requirements, you should obtain the capability to print other predefined files. Also, you can consider the addition of one or more replaceable parameters to the previously presented programs. This would enable you to select different files for printing by passing the filename(s) as replaceable parameters to the PRINT command in your batch file.

LABEL GENERATOR

Many people need to create mailing labels for repetitive correspondence. A large mailing label program is simply not economically feasible for a small number of names and addresses, so here we explore a simple way to create labels. Because a list of names and addresses can constantly change, it is important to allow for easy changes to those parameters.

You can control the printer from the keyboard. If you issue the command

```
COPY CON: PRN:
```

everything typed in at the keyboard will go to the printer. To try this, simply enter this command line at the DOS prompt and then begin typing simple labels with a name, followed by an address, followed by the city, state, and zip code. Leave a few lines between each label by simply pressing the RETURN or ENTER key. Notice that as you type each character the screen shows what you are typing, but the printer does nothing. This is because the computer and printer are storing the keystrokes in a memory area called a *buffer.* The buffer is flushed—that is, sent to the selected device (PRN:)—when it becomes full or when you signify that the operation is finished.

When you have typed three or four labels, press and hold the CTRL key and, while holding it down, press the Z. Next, press the ENTER or RETURN key. The CTRL-Z keystroke sequence is used to tell DOS that the COPY command is completed. The printer will then print the labels stored in the buffer, because the completion of the COPY command causes its contents to be flushed.

As an alternative to using the DOS COPY command, you could create a batch file to issue the COPY command and advise the user when the label maker is ready. Program Listing 6.3 shows an example of a batch file called MAKELABL.BAT. Line 1 turns off the echo function so you won't see each command as it is executed. Line 2 clears the screen, and lines 3 through 11 tell the user what the command MAKELABL does. Notice that each ECHO command has a period, or dot, after it. This is simply to keep the ECHO command from telling us whether or not echo is on or off. The ECHO command, without anything after it, will determine whether echo is on or off; all we want to do here is make the screen display attractive, so the period or some other character is needed. Also notice that line 9 tells you what to do when you have finished typing labels. Line 12 contains the COPY command, which directs the computer to copy everything entered from the keyboard (CON) to the printer (PRN).

Program Listing 6.3

MAKELABL.BAT

```
 1:*echo off
 2: cls
 3: echo .
 4: echo .
 5: echo MAKING LABELS....Be sure printer is loaded with labels
 6: echo .
 7: echo .         .........begin typing names and addresses.....
 8: echo .
 9: echo .         ...when finished, press 'CTRL' key and letter 'Z'...
10: echo .
11: echo .
12: copy con prn
```

Following is a sample run of the MAKELABL batch program as it appears on the screen. The user simply entered the command MAKELABL, created three labels, and pressed CTRL-Z.

```
MAKING LABELS....Be sure printer is loaded with labels
.
.         .........begin typing names and addresses.....
.
.         ...when finished, press 'CTRL' key and letter 'Z'...
.
.
```

```
Tom Dolittle
123 West Main Street
Atlanta, GA  30334

Mary Smith
64 Peachtree Avenue
Atlanta, GA  30330


Don Miller
1654 Northlake Drive
Marietta, GA  31243
^Z
        1 File(s) copied

C>
```

Next you can see the printed output created by running the MAKELABL batch file. You may need a little practice to space the labels correctly if you are using pregummed labels in the printer. Another, more reliable method is to prepare the labels on plain paper, cut them out with a pair of scissors, and glue them to the appropriate envelopes—or better yet, print them on plain paper and photocopy them onto sheets of labels.

```
Tom Dolittle
123 West Main Street
Atlanta, GA  30334

Mary Smith
64 Peachtree Avenue
Atlanta, GA  30330
```

PRINTER CONTROL

```
Don Miller
1654 Northlake Drive
Marietta, GA  31243
```

Alternative Program

If you need only a few labels each month, and the addresses differ from month to month, the previously described method for directing console entries to the printer is satisfactory. However, if you need a large number of names and addresses each month, the method just described is impractical. We will therefore modify our label generator to use the file storage capability of our computer.

Instead of directly routing data from the console (CON) to the printer (PRN), you could create one or more files that contain the information you wish to print as labels. Then, as your need for labels arises you could copy the appropriate files to the printer.

Suppose you wish to construct a file of names and addresses for Christmas cards. You could create a file named XMAS.LST from the console by entering the following DOS command:

```
COPY CON: XMAS.LST
```

Then, after you finish entering the appropriate names and addresses you would save the file by entering CTRL-Z. A better way to create the file would be to use the EDLIN text editor (or a word processor), because you could then edit previous entries.

Once you've constructed your file, you can again use the DOS COPY command, this time using the command to route the contents of the file to the printer. Thus, you could enter the command

```
COPY XMAS.LST PRN:
```

to print your Christmas mailing list.

PRINTER MODE CONTROL

Most dot matrix printers change their print modes when a program transmits predefined codes to the printers. Although some recently introduced printers include a panel control to select some of the available print modes, a majority of the print modes must still be set by the transmission of an appropriate control code from the computer. A mechanism that makes it easy for you to change printer modes of operation can therefore be a valuable productivity tool.

In this section we will examine methods of printer control. The examples will use the Epson control codes to set different modes of operation on an Epson LQ-1500 printer. By examining the manual for your printer, you can substitute control codes appropriate to your printer so the examples will work on your equipment.

Control Codes

Table 6.1 lists the control codes that enable and disable the different print modes of the Epson LQ-1500. The expanded-print mode results in characters twice as wide as the standard character width. In the compressed-print mode, the Epson prints 17.16 characters per inch instead of its normal 10 characters per inch, so that you get approximately 70 percent more characters on a line. In the emphasized mode, characters are struck twice with a minimal horizontal space between strikes, resulting in boldface appearance of the characters. The Epson LQ-1500's doublestrike mode of operation also causes characters to be struck twice; however, the second printing is slightly shifted to eliminate the dots in the character. This print mode is sometimes known as near letter-quality (NLQ) printing.

Table 6.1a

Epson LQ-1500 Printer Control Codes—Proportional Print Styles

	Pica Master Select Number	Expanded Master Select Number
Regular	2	34
Emphasized	10	42
Double-strike	18	50
Double-strike emphasized	26	58
Italic		
Regular	66	98
Emphasized	74	106
Double-strike	82	114
Double-strike emphasized	90	122
Underlined		
Regular	130	162
Emphasized	138	170
Double-strike	146	178
Double-strike emphasized	154	186
Italic underlined		
Regular	194	226
Emphasized	202	234
Double-strike	210	242
Double-strike emphasized	218	250

Table 6.1b

Epson LQ-1500 Printer Control Codes—Letter Quality Print Styles

	Pica Master Select Number	Pica Expanded Master Select Number	Elite Master Select Number	Elite Expanded Master Select Number
Regular	0	32	1	33
Emphasized	8	40	9	41
Double-strike	16	48	17	49
Double-strike emphasized	24	56	25	57
Italic				
Regular	64	96	65	97
Emphasized	72	104	73	105
Double-strike	80	112	81	113
Double-strike emphasized	88	120	89	121
Underlined				
Regular	128	160	129	161
Emphasized	136	168	137	169
Double-strike	144	176	145	177
Double-strike emphasized	152	184	153	185
Italic underlined				
Regular	192	224	193	225
Emphasized	200	232	201	233
Double-strike	208	240	209	241
Double-strike emphasized	216	248	217	249

Table 6.1c

Epson LQ-1500 Printer Control Codes—Draft Pica Print Styles

	Pica Master Select Number	Compressed Master Select Number	Expanded Master Select Number	Compressed Expanded Master Select Number
Regular	0	4	32	36
Emphasized	8	12	40	44
Double-strike	16	20	48	52
Double-strike emphasized	24	28	56	60
Italic				
Regular	64	68	96	100
Emphasized	72	76	104	108
Double-strike	80	84	112	116
Double-strike emphasized	88	92	120	124
Underlined				
Regular	128	132	160	164
Emphasized	136	140	168	172
Double-strike	144	148	176	180
Double-strike emphasized	152	156	184	188
Italic underlined				
Regular	192	196	224	228
Emphasized	200	204	232	236
Double-strike	208	212	240	244
Double-strike emphasized	216	220	248	252

Table 6.1d

Epson LQ-1500 Printer Control Codes—Draft Elite Print Styles

	Pica Master Select Number	Compressed Master Select Number	Expanded Master Select Number	Compressed Expanded Master Select Number
Regular	1	5	33	37
Emphasized	9	13	41	45
Double-strike	17	21	49	53
Double-strike emphasized	25	29	57	61
Italic				
Regular	65	69	97	101
Emphasized	73	77	105	109
Double-strike	81	85	113	117
Double-strike emphasized	89	93	121	125
Underlined				
Regular	129	133	161	165
Emphasized	137	141	169	173
Double-strike	145	149	177	181
Double-strike emphasized	153	157	185	189
Italic underlined				
Regular	193	197	225	229
Emphasized	201	205	233	237
Double-strike	209	213	241	245
Double-strike emphasized	217	221	249	253

Control Through BASIC

You can call a BASIC program that provides a menu of available printing options and initiate the transmission of appropriate control codes to the printer. When you use this method of printer mode control, you can run the BASIC program directly from the command line or indirectly from a batch file. You could, for example, use a batch file to call the BASIC program before you ran a word processor that did not support the printer. In this manner you can provide your own printer support, regardless of the capability of the application program.

Assuming that the name of the BASIC printer-control program is PRINTER.BAS, the following batch file first initiates the BASIC program. Then, it changes the current directory to the directory named WP and executes the word processing program whose executable file is also named WP. When you exit from the word processing program to return to DOS, the batch file will return control to the root directory.

```
COPY CON: WP.BAT
BASICA PRINTER.BAS
CD\WP
WP
CD\
^Z
```

Program Listing 6.4 contains a listing of the BASIC program written to provide a menu of printer mode option selections. Note that due to the interactive nature of the BASIC language we are able to advise the computer user of illegal entries and permit the user to reenter the desired printer mode and status.

Program Listing 6.4

Epson LQ-1500 Utility Program

```
20 '**    UTIL03.BAS - LQ-1500 PRINTER SETUP UTILITY PROGRAM              **
30 '**                 Written by Tom Dolittle                            **
40 '**                 August 9, 1984                                     **
50 '**                                                                    **
60 '**    This program is designed to allow the user to select various    **
70 '**    print styles for the output to the EPSON LQ-1500 printer. This  **
80 '**    program will initialize the printer for the output print style  **
90 '**    selected by the user, and offer to show him or her a sample.    **
100'*************************************************************************
110'
120 KEY OFF: PS=0: PLAY "CEGC"           'PS=0 indicates no sample printed
130 SAMPLE$="Sample EXAMPLE to Demonstrate Print Style !"
150 CLS:COLOR 15,0
160 PRINT "***** EPSON LQ-1500 PRINTER SETUP UTILITY *****"
170 PRINT "-----------------------------------------------"
180 COLOR 7,0
190 PRINT
200 PRINT "        1 - Proportional Print Styles"
210 PRINT "        2 - Letter Quality Print Styles"
220 PRINT "        3 - Draft Pica Print Styles"
230 PRINT "        4 - Draft Elite Print Styles"
240 PRINT "        5 - Print a Sample of Current Settings"
250 PRINT "        6 - Exit"
260 LOCATE 11,1
270 MIN=1:MAX=6:GOSUB 2070
280 ON I GOTO 370,660,1070,1480,1890,1950
290 PRINT:PRINT "Would you like to see a sample printout ???...(Y/N)...";
300 INPUT Y$
310 IF Y$="Y" OR Y$="y" THEN 1900
320 GOTO 150
370 '-----------------------------------------------------------------------
380 '  Proportional Print Styles....
390 '
400 COLOR 15,0
410 CLS:PRINT "PROPORTIONAL PRINT STYLES"
420 PRINT "-------------------------"
430 COLOR 7,0
440 PRINT
450 PRINT "1 - Pica Regular                    5 - Expanded Regular"
460 PRINT "2 - Pica Emphasized                 6 - Expanded Emphasized"
470 PRINT "3 - Pica Doublestrike               7 - Expanded Doublestrike"
480 PRINT "4 - Pica Doublestrike Emphasized    8 - Expanded Doublestrike Emphasized"
```

```
490 PRINT:MIN=1:MAX=8:GOSUB 2070:A=I
500 PRINT
510 GOSUB 1990                  'go ask user for option selection
520 B=I-1                       'B holds the option selected (less 1)
530 C=B*64                      'C is the offset for option selection
540 ON A GOTO 550,560,570,580,590,600,610,620
550 N=2:GOTO 630
560 N=10:GOTO 630
570 N=18:GOTO 630
580 N=26:GOTO 630
590 N=34:GOTO 630
600 N=42:GOTO 630
610 N=50:GOTO 630
620 N=58:GOTO 630
630 N=N+C                    'add offset for option selection
640 LPRINT CHR$(27) "!" CHR$(N);
650 GOTO 290
660 '--------------------------------------------------------------------------
670 '  Letter Quality Print Styles....
680 '
690 COLOR 15,0
700
710
720 COLOR 7,0
730 PRINT
740 PRINT "1 - Pica Regular                  9 - Pica Expanded Regular"
750 PRINT "2 - Pica Emphasized              10 - Pica Expanded Emphasized"
760 PRINT "3 - Pica Doublestrike            11 - Pica Expanded Doublestrike"
770 PRINT "4 - Pica Doublestrike Emphasized 12 - Pica Expanded Doublestrike Emphasized"
780 PRINT "5 - Elite Regular                13 - Elite Expanded Regular"
790 PRINT "6 - Elite Emphasized             14 - Elite Expanded Emphasized"
800 PRINT "7 - Elite Doublestrike           15 - Elite Expanded Doublestrike"
810 PRINT "8 - Elite Doublestrike Emphasized 16 - Elite Expanded Doublestrike Emphasized"
820 PRINT:MIN=1:MAX=16:GOSUB 2070:A=I
830 PRINT
840 GOSUB 1990                  'go ask user for option selection
850 B=I-1                       'B holds the option selected (less 1)
860 C=B*64                      'C is the offset for option selection
870 ON A GOTO 880,890,900,910,920,930,940,950,960,970,980,990,1000,1010,1020,1030
880 N=0:GOTO 1040
890 N=8:GOTO 1040
900 N=16:GOTO 1040
910 N=24:GOTO 1040
```

```
920 N=1:GOTO 1040
930 N=9:GOTO 1040
940 N=17:GOTO 1040
950 N=25:GOTO 1040
960 N=32:GOTO 1040
970 N=40:GOTO 1040
980 N=48:GOTO 1040
990 N=56:GOTO 1040
1000 N=33:GOTO 1040
1010 N=41:GOTO 1040
1020 N=49:GOTO 1040
1030 N=57:GOTO 1040
1040 N=N+C                 'add offset for option selection
1050 LPRINT CHR$(27) "x" CHR$(1); CHR$(27) "!" CHR$(N);
1060 GOTO 290
1070 '------------------------------------------------------------------------
1080 '  Draft Pica Print Styles....
1090 '
1100 COLOR 15,0
1110 CLS:PRINT "DRAFT PICA PRINT STYLES"
1120 PRINT "-----------------------"
1130 COLOR 7,0
1140 PRINT
1150 PRINT "1 - Pica Regular                    9 - Expanded Regular"
1160 PRINT "2 - Pica Emphasized                10 - Expanded Emphasized"
1170 PRINT "3 - Pica Doublestrike              11 - Expanded Doublestrike"
1180 PRINT "4 - Pica Doublestrike Emphasized   12 - Expanded Doublestrike Emphasized"
1190 PRINT "5 - Compressed Regular             13 - Compressed Expanded Regular"
1200 PRINT "6 - Compressed Emphasized          14 - Compressed Expanded Emphasized"
1210 PRINT "7 - Compressed Doublestrike        15 - Compressed Expanded Doublestrike"
1220 PRINT "8 - Compressed Doublestrike Emph.  16 - Compressed Expanded Doublestrike Emph."
1230 PRINT:MIN=1:MAX=16:GOSUB 2070:A=I
1240 PRINT
1250 GOSUB 1990               'go ask user for option selection
1260 B=I-1                    'B holds the option selected (less 1)
1270 C=B*64                   'C is the offset for option selection
1280 ON A GOTO 1290,1300,1310,1320,1330,1340,1350,1360,1370,1380,1390,1400,1410,1420,1430,1440
1290 N=0:GOTO 1450
1300 N=8:GOTO 1450
1310 N=16:GOTO 1450
1320 N=24:GOTO 1450
1330 N=4:GOTO 1450
1340 N=12:GOTO 1450
```

```
1350 N=20:GOTO 1450
1360 N=28:GOTO 1450
1370 N=32:GOTO 1450
1380 N=40:GOTO 1450
1390 N=48:GOTO 1450
1400 N=56:GOTO 1450
1410 N=36:GOTO 1450
1420 N=44:GOTO 1450
1430 N=52:GOTO 1450
1440 N=60:GOTO 1450
1450 N=N+C                 'add offset for option selection
1460 LPRINT CHR$(27) "x" CHR$(0); CHR$(27) "!" CHR$(N);
1470 GOTO 290
1480 '-----------------------------------------------------------------------
1490 '  Draft Elite Print Styles....
1500 '
1510 COLOR 15,0
1520 CLS:PRINT "DRAFT ELITE PRINT STYLES"
1530 PRINT "------------------------"
1540 COLOR 7,0
1550 PRINT
1560 PRINT "1 - Pica Regular                    9 - Expanded Regular"
1570 PRINT "2 - Pica Emphasized                10 - Expanded Emphasized"
1580 PRINT "3 - Pica Doublestrike              11 - Expanded Doublestrike"
1590 PRINT "4 - Pica Doublestrike Emphasized   12 - Expanded Doublestrike Emphasized"
1600 PRINT "5 - Compressed Regular             13 - Compressed Expanded Regular"
1610 PRINT "6 - Compressed Emphasized          14 - Compressed Expanded Emphasized"
1620 PRINT "7 - Compressed Doublestrike        15 - Compressed Expanded Doublestrike"
1630 PRINT "8 - Compressed Doublestrike Emph.  16 - Compressed Expanded Doublestrike Emph."
1640 PRINT:MIN=1:MAX=16:GOSUB 2070:A=I
1650 PRINT
1660 GOSUB 1990               'go ask user for option selection
1670 B=I-1                    'B holds the option selected (less 1)
1680 C=B*64                   'C is the offset for option selection
1690 ON A GOTO 1700,1710,1720,1730,1740,1750,1760,1770,1780,1790,1800,1810,1820,1830,1840,1850
1700 N=1:GOTO 1860
1710 N=9:GOTO 1860
1720 N=17:GOTO 1860
1730 N=25:GOTO 1860
1740 N=5:GOTO 1860
1750 N=13:GOTO 1860
1760 N=21:GOTO 1860
1770 N=29:GOTO 1860
```

```
1780 N=33:GOTO 1860
1790 N=41:GOTO 1860
1800 N=49:GOTO 1860
1810 N=57:GOTO 1860
1820 N=37:GOTO 1860
1830 N=45:GOTO 1860
1840 N=53:GOTO 1860
1850 N=61:GOTO 1860
1860 N=N+C                   'add offset for option selection
1870 LPRINT CHR$(27) "x" CHR$(0); CHR$(27) "!" CHR$(N);
1880 GOTO 290
1890 '-------------------------------------------------------------------
1900 '  Print a Sample of Current Settings
1910 '
1920 LPRINT SAMPLE$
1930 LPRINT SAMPLE$
1935 PS=1                                  'print sample switch = Yes
1940 GOTO 1050
1950 '-------------------------------------------------------------------
1960 '  End of Program...return to SYSTEM....
1970 IF PS<>0 THEN LPRINT CHR$(12)        'if sample printed...give form feed
1980 SYSTEM
1990 '------------- Option Selection Common to All Styles ---------
2000 PRINT "Options are :      1 - Regular Print"
2010 PRINT "                   2 - Italic"
2020 PRINT "                   3 - Underlined"
2030 PRINT "                   4 - Italic Underlined"
2040 PRINT
2050 MIN=1:MAX=4:GOSUB 2070
2060 RETURN
2070 '-------------------------------------------------------------------
2080 '  Subroutine to ensure user enters a proper value...allowable entries
2090 '  are passed to this routine in the variables MIN and MAX, and the
2100 '  user must select a number within this range....
2110 INPUT "Enter the NUMBER of your selection...";I
2120 IF I<MIN OR I>MAX THEN 2140
2130 RETURN
2140 PRINT "*** ERROR *** You MUST enter a NUMBER from above...try again..."
2150 BEEP:BEEP:GOTO 2110
```

Lines 10 through 110 are comments and information about the program itself. The single apostrophe (') is an abbreviation for the

BASIC REM statement and makes the rest of the line after the apostrophe a comment which explains an operation but which is ignored by the BASIC interpreter (or compiler).

```
10 '*********************************************************************
20 '**    UTIL03.BAS - LQ-1500 PRINTER SETUP UTILITY PROGRAM            **
30 '**                 Written by Tom Dolittle                          **
40 '**                 August 9, 1984                                   **
50 '**                                                                  **
60 '**    This program is designed to allow the user to select various  **
70 '**    print styles for the output on the EPSON LQ-1500 printer. This **
80 '**    program will initialize the printer for the output print style **
90 '**    selected by the user, and offer to show him/her a sample.     **
100'*********************************************************************
110'
```

Line 120 sets a variable (PS) to indicate later in the program whether or not a sample has been printed. The PLAY command also plays a little musical tune to introduce the program. You don't need the music for the program to function properly, but it's just an added touch. Line 130 sets the BASIC variable SAMPLE$ to the string value to be printed when an example is requested.

```
120 KEY OFF: PS=0: PLAY "CEGC" 'PS=0 indicates no sample printed
130 SAMPLE$="Sample EXAMPLE to Demonstrate Print Style !"
```

Line 150 begins the first menu by clearing the screen and setting COLOR options for the following PRINT commands. This particular COLOR command simply sets the next lines printed on the screen into highlighted mode. Lines 160 through 250 place the first menu on the screen with simple PRINT commands. Note that line 180 returns the selections to unhighlighted mode, although the title remains highlighted. Line 260 locates the cursor at position 1 of line 11 on the screen.

```
150 CLS:COLOR 15,0
160 PRINT "***** EPSON LQ-1500 PRINTER SETUP UTILITY *****"
170 PRINT "----------------------------------------------"
180 COLOR 7,0
190 PRINT
200 PRINT "        1 - Proportional Print Styles"
210 PRINT "        2 - Letter Quality Print Styles"
220 PRINT "        3 - Draft Pica Print Styles"
230 PRINT "        4 - Draft Elite Print Styles"
240 PRINT "        5 - Print a Sample of Current Settings"
250 PRINT "        6 - Exit"
260 LOCATE 11,1
```

Line 270 calls a subroutine to get your input. The values you enter cannot be less than 1 or greater than 6, because 1 through 6 are the options shown on the menu. Line 270 first sets the variable MIN to the minimum allowable value of 1 and the variable MAX to the maximum allowable value of 6 and then calls the input subroutine starting at line 2070. This subroutine will be discussed later, but it simply ensures that you type in a valid response (that is, a number between MIN and MAX).

```
270 MIN=1:MAX=6:GOSUB 2070
```

Line 280 uses the BASIC ON . . . GOTO statement to transfer control to the part of the program that handles the particular function selected from the menu. If you select option 1 from the menu, the variable I will contain the value 1 (when it returns from the subroutine at 2070), and the BASIC command would be `ON 1 GOTO 370` because 370 is the first entry in the GOTO part of the command. If you select 2 from the menu, the BASIC command would be `ON 2 GOTO 660`, because 660 is the second entry in the ON. .GOTO command. Similarly, a selection of 3 would result in ON 3 GOTO 1070, and so on. Each of these routines will be explained later, but as each routine is completed it returns control to the BASIC command at line 290.

```
280 ON I GOTO 370,660,1070,1480,1890,1950
```

Lines 290 through 310 ask if you would like to see a sample of the type of print just selected. This sample capability will give you a chance to see what the output will look like prior to printing an actual document. If you answer with a Y or y, control is passed to a subroutine (at line 1900) that prints a sample on the printer. Line 320 next returns control to line 150, which clears the screen and places the first option menu up for you to select an option. At this point, you can reset the printer to another setting or exit from the program with the printer set for the mode you selected.

```
290 PRINT:PRINT "Would you like to see a sample printout ???. . .(Y/N). . .";
300 INPUT Y$
310 IF Y$="Y" OR Y$="y" THEN 1900
320 GOTO 150
```

Because the Epson LQ-1500 can print four different styles of print, four of the selections on the first menu are for each of these styles and a separate routine is called for each different selection. The routine for selection 1 is shown in lines 370 through 650. Lines 370 through 390 are simply remarks and documentation. Lines 400 through 480 clear the screen, place the heading at the top in emphasized type, and display the options for proportional print styles. Line 490 sets the MIN and MAX values for possible answers and calls the subroutine at line 2070 for you to select a number. When the subroutine returns, the number selected comes back in the variable I and is saved in a new variable called A. This value (selection) is saved because the subroutine at 2070 will be used again and the value in I will be changed.

```
370 '-------------------------------------------------------------------
380 '  Proportional Print Styles....
390 '
400 COLOR 15,0
410 CLS:PRINT "PROPORTIONAL PRINT STYLES"
420 PRINT "------------------------"
430 COLOR 7,0
440 PRINT
450 PRINT "1 - Pica Regular                    5 - Expanded Regular"
460 PRINT "2 - Pica Emphasized                 6 - Expanded Emphasized"
470 PRINT "3 - Pica Doublestrike               7 - Expanded Doublestrike"
480 PRINT "4 - Pica Doublestrike Emphasized   8 - Expanded Doublestrike Emphasized"
490 PRINT:MIN=1:MAX=8:GOSUB 2070:A=I
```

Line 510 calls a subroutine at 1990 that provides the capability to select additional options common to all four print styles. Each style can print in regular, italic, underlined, and italic underlined modes. The control codes are listed in Table 6.1 for the Epson LQ-1500.

The subroutine at lines 1990 through 2060 enables you to select the desired mode and returns to the command immediately after the `GOSUB 1990`. In this case the program control would return to line 520, but this subroutine is used by the three other style routines when appropriate.

```
 500 PRINT
 510 GOSUB 1990                        'go ask user for option selection

1990 '--------------Option Selection Common to All Styles ---------
2000 PRINT "Options are :       1 - Regular Print"
2010 PRINT "                    2 - Italic"
2020 PRINT "                    3 - Underlined"
2030 PRINT "                    4 - Italic Underlined"
2040 PRINT
```

```
2050 MIN=1:MAX=4:GOSUB 2070
2060 RETURN
```

Line 520 set the variable B equal to the value selected in the subroutine at 1990 less 1. Line 530 next multiplies the value in variable B by 64, placing the product in variable C. There is a reason for this that will become apparent shortly. Line 540 uses the BASIC ON...GOTO command again to transfer program control based upon the value you selected in the second menu. In our example, assume you selected option 1 on the first menu (`1 - Proportional Print Styles`) and option 6 on the second menu (`6 - Expanded Emphasized`). Because the variable A contains the value 6, the BASIC statement becomes `ON 6 GOTO 600`, because 600 is the sixth entry after the `ON...GOTO` statement. Lines 550 through 620 each set a variable called N to a particular value, then transfer control to line 630.

```
520 B=I-1     'B holds the option selected (less 1)
530 C=B*64    'C is the offset for option selection
540 ON A GOTO 550,560,570,580,590,600,610,620
550 N=2:GOTO 630
560 N=10:GOTO 630
570 N=18:GOTO 630
580 N=26:GOTO 630
590 N=34:GOTO 630
600 N=42:GOTO 630
610 N=50:GOTO 630
620 N=58:GOTO 630
```

A close look at the values assigned in lines 550 through 620 indicates that they correspond to the entries in the Proportional Print Styles entry in Table 6.1. Notice that the Master Select Number for Pica Regular is 2 (same as in line 550), the Master Select Number for Pica Emphasized is 10 (same as in line 560), and so on. Therefore, the variable N contains the Master Select Code for regular print within the Proportional Print Styles available. We selected `6 - Expanded Emphasized` from Menu 2, so our value for N here would be 42 (line 600).

Line 630 next takes the value in the variable N (42 in our sample) and adds the value in the variable C. Remember that the value in C came from a B variable obtained in a subroutine at line 1990, was lowered by 1 and was multiplied by 64. Note in the subroutine in lines 1990 through 2060 that you can select among regular print, italic, underlined, or italic underlined. A glance at the entries for Proportional Print Styles in Table 6.1 also shows these four options and the Master Select Codes for each. Suppose we chose italic print for this example. Then the number 2 would have been returned in the variable I, 2 minus 1 is 1 (B now equals 1 in line 520) and 1*64 equals 64 (line 530). The C variable now contains the value 64.

Therefore, the value of N at line 630 becomes N = 42 + 64, or 106. Looking back at the Proportional Print Styles table we can see that the Master Select Code 106 corresponds to Proportional (first menu selection 1) Expanded Emphasized (second menu selection 6) Italic (subroutine 1990 through 2060 which was called at line 510). A further examination of the Print Style tables reveals that the difference in value between regular and italic is always 64 in the Master Select Codes for each type of print desired. This pattern can be used easily to program for the other tables in subroutines for Letter Quality, Draft Pica, and Draft Elite Print Styles.

```
630 N=N+C
```

Finally, line 640 uses the LPRINT command to send control codes to the printer to set it to the desired mode. The CHR$(27) is an escape character used quite often to get the attention of a printer to be reset. The ! character is used strictly by the Epson LQ-1500 with the escape to signify that a Master Select Code will follow, and the CHR$(N) is the Master Select Code for the desired printer setting. The semicolon at the end of the line prevents the printer from advancing one line; most printers automatically advance after an LPRINT command that is terminated by a return, because the LPRINT command sends a carriage return and line feed automatically. However, if you terminate the command with a semicolon, printing is delayed until the next LPRINT that is terminated with a return.

```
                               'add offset for option selection
640 LPRINT CHR$(27) "!" CHR$(N);
650 GOTO 290
```

Remember that the Master Select Code is unique to the Epson LQ-1500, but that almost all dot matrix printers and many letter-quality printers can change print styles using special control codes.

A discussion of the other routines in the program for selecting Letter Quality, Draft Pica, and Draft Elite Print Styles is unnecessary, because they are similar to the routine for Proportional Print Styles. A look at lines 1890 through 1940 shows that a subroutine in lines 290 through 310 prints a sample, if you request it.

```
1890 '-------------------------------------------------------------
1900 '  Print a Sample of Current Settings
1910 '
1920 LPRINT SAMPLE$
1930 LPRINT SAMPLE$
1935 PS=1                               'print sample switch = Yes
1940 GOTO 150
```

Lines 1920 and 1930 simply print the characters in the variable SAMPLE$, which was initially loaded at line 130. Line 1935 then resets the variable PS to the value 1 to indicate that a sample has been printed. Line 1940 returns control to line 150 of the program so you can change settings if the sample is not what you wanted.

Selecting option 6 from the first menu exits the program. The program is completed by cleaning itself up at lines 1950 through 1980. Line 1970 checks to see if the print sample variable is set to some value other than zero. If a sample has been printed, then the LPRINT CHR$(12) is performed to send a form feed to the printer. Because the sample is printed on the sheet of paper currently in the printer, a new and clean sheet should be placed in the printer so that our printout will start at the top of the page and on clean paper. Line 1980 then returns control to the operating system.

```
1950 '-----------------------------------------------------------------------
1960 '  End of Program...return to SYSTEM....
1970 IF PS<>0 THEN LPRINT CHR$(12)  'if sample printed...give form feed
1980 SYSTEM
```

The subroutine at lines 1070 through 2150 ensures that you input a valid value. In other words, if the program offers a menu that asks you to select a number between 1 and 6, the program should prevent you from entering the number 55 or any other number not on the menu.

Lines 2070 through 2100 simply document what the routine does. Line 2110 uses the INPUT command to prompt you to enter a number and waits for the response. The number you select is placed in the variable I. Line 2120 then checks the number in the variable I to ensure that it is neither less than the minimum value nor more than the maximum value desired by the program menu. If the number is within the MIN and MAX value range, control goes to line 2130, which returns control to the command after the GOSUB command that called this subroutine. If the value entered in I is outside the allowable range, line 2140 is executed to tell you that you must enter a number from the menu. Line 2150 uses the BEEP command to make you aware of an error and transfers control back to line 2110 for you to try again. This is a simple but effective method of checking input validity.

```
2070 '--------------------------------------------------------------------
2080 '  Subroutine to ensure user enters a proper value...allowable entries
2090 '  are passed to this routine in the variables MIN and MAX, and the
2100 '  user must select a number within this range....
2110 INPUT "Enter the NUMBER of your selection...";I
2120 IF I<MIN OR I>MAX THEN 2140
2130 RETURN
2140 PRINT "*** ERROR *** You MUST enter a NUMBER from above...try again..."
2150 BEEP:BEEP:GOTO 2110
```

An Alternative Printer Control Method

If your only requirement is to set a few printer modes of operation, you can consider the direct use of DOS, eliminating the previously described method that incorporated a BASIC language program. Here the key to the exclusive use of DOS is the ability of the operating system to directly send printer control characters to your printer.

You can send any printer control character from DOS to your printer by pressing the ALT key and the numeric keypad keys. Simply press and hold down the ALT key while you enter the numeric value of the ASCII code for the character you desire. As an example, the decimal value of the escape character is 027. Thus, you would enter an escape character into a batch file by pressing and holding the ALT key while you enter the decimal value 027 from the numeric keypad.

Because the first 32 ASCII characters are defined to perform control functions a table containing their decimal values, mnemonic designation and meaning is presented in A DOS Database in Chapter

7, which describes the use of the ALT key for generating audio and graphics through DOS. For a full list of the decimal values of the ASCII character set, check the appendixes of your BASIC language manual.

If you attempt to enter the escape code whose ASCII value is 027, you will note that DOS will interpret that character as a request to stop accepting data. In effect, ASCII 027 is interpreted literally as a request to escape from performing a current operation. Thus, DOS not only stops accepting data, but it also terminates the current line entry and resets itself to accept a new command line entry. Due to this escape interpretation, you cannot directly enter a conventional escape character that has an ASCII value of 027. Fortunately, you can fool DOS into accepting a disguised escape code. This is accomplished by adding the value of 128 to 27 for a new value of ASCII 155. Here the addition of decimal 128 is equivalent to changing the high-order eighth bit of the ASCII escape character from a 0 to a 1. When you enter ASCII 155, DOS interprets that character as a non-escape character code.

Now that you have a method of entering a disguised escape character, what is its effect when it is sent to the printer?

Printer Operations

Most printers only use the first seven bits of a character when they operate in a normal ASCII print mode in which the character set is assumed to range in value from decimal 0 through decimal 127. When operating in this print mode, the printer ignores the setting of the eighth bit, resulting in ASCII 27 and ASCII 155 both being interpreted as the escape character. Thus, the ALT 155 key combination is both acceptable to DOS and usable to generate an escape character to the printer.

In this section's examples, we will assume that an Epson or Epson compatible printer is connected to our computer. Thus, we will examine the creation of several batch files that use Epson printer control codes to set different printer modes of operation. If your printer does not support these control codes, you can examine your printer manual to determine the appropriate codes to use in the examples presented in this section.

In Table 6.2 you will find a list of the ASCII character or characters, their decimal values, and resulting functions you can use with an Epson printer to enable or disable several popular printing modes of operation.

The condensed print mode changes the character size to approximately 60 percent of the width of normal characters, permitting more characters to be printed on a line. Normally, this print mode

Table 6.2

Popular Epson Print Modes

ASCII Character(s)	Decimal Value(s)	Function
SI	15	Select Condensed Mode
DC2	18	Cancel Condensed Mode
SO	14	Select Double-Wide Mode (one line)
DC4	20	Cancel Double-Wide Mode (one line)
ESC W	27 87	Turn Double-Wide Mode On/Off
ESC E	27 69	Select Emphasized Mode
ESC F	27 70	Cancel Emphasized Mode
ESC G	27 71	Select Double-Strike Mode
ESC H	27 72	Cancel Double-Strike Mode

is used for printing out a long database record or a spreadsheet listing. As indicated in Table 6.2, condensed mode is selected by sending the SI character whose decimal value is 15 to the printer. By sending the DC2 character whose decimal value is 18 this print mode can be canceled.

The second grouping of print modes in Table 6.2 covers double-wide printing. As the name of the mode implies, double-wide printing doubles the width of all characters. The ASCII SO character, decimal 14, is used to select the printing of one line of data in double-wide mode. Similarly, the ASCII DC4 control character, decimal 20, is used to cancel double-wide mode printing. The third member of this grouping can be viewed as a toggle switch, because each occurrence turns the double-wide mode on or off, reversing the previous setting. This switch is activated by the ASCII ESC W code sequence whose decimal value is 27 87.

The last two print groupings in Table 6.2 control enabling and disabling emphasized and doublestrike printing. Emphasized printing makes text bolder looking by printing the dots that make up each character twice, placing the second dot of each character slightly to the right of the first dot. Doublestrike mode printing can also be used to make text bolder. In this print mode each line is printed twice, with the second printing occurring slightly *below* the first.

Compressed Printing

Let us assume you desire to create a batch file that can be used to print text in a compressed mode. You could create the batch file named COMPRESS.BAT, which is shown in Program Listing 6.5.

Program Listing 6.5 **COMPRESS.BAT**

```
ECHO <ALT>15>LPT1:
TYPE %1>LPT1:
ECHO <ALT>18>LPT1:
```

Program Listing 6.5 uses the symbol <ALT> to denote that the ALT key was pressed while the numeric value following the symbol was entered from the numeric keypad. Here ECHO <ALT> 15 will generate the ASCII character whose decimal value is 15. When this character is routed to the printer it turns on compressed printing, which is also known as *condensed printing.* The second line in the COMPRESS.BAT file will route an ASCII text file assigned to the %1 replaceable parameter to the printer, and the third line in the file disables compressed printing.

As an alternative to using the ALT key, you can enter many printer control codes directly by using the CTRL key and an uppercase letter. As an example, consider the COMPRESS.BAT program listed in Program Listing 6.5. Because <ALT> 15 is a CTRL-O, and <ALT> 18 is a CTRL-R, you can enter those control characters directly. When you do so, the circumflex accent (caret or ^) preceding the letter on the screen signifies that the letter is a control character. Program Listing 6.6 illustrates the direct entry of control characters into the COMPRESS.BAT file to enable and disable compressed printing.

Program Listing 6.6 **Direct Entry of Control Characters**

```
ECHO ^O >LPT1:
TYPE %1 >LPT1:
ECHO ^R >LPT1:
```

To illustrate the usefulness of COMPRESS.BAT, let us assume you have a file created with EDLIN that has one or more lines of text that exceed 80 columns in length. The following shows the contents of the file SAMPLE.TXT, which meets the preceding criteria.

```
1:*The quick Brown Fox jumped over the lazy dog who was sunbathing in
the shade when the moon was bright.
```

Now we can use the previously created COMPRESS.BAT file to print the sample text file. To do so, you would enter the command

```
COMPRESS SAMPLE.TXT
```

Issuing that command invokes the file COMPRESS.BAT, with SAMPLE.TXT assigned to the %1 replaceable parameter. This causes the printer to switch to a compressed print mode, the contents of SAMPLE.TXT to print, and the compressed print mode to be disabled to permit resumption of normal printing. As a result of the preceding operations, the two-line EDLIN created file is printed.

```
The quick Brown Fox jumped over the lazy dog who was sunbathing in
the shade when the moon was bright.
```

Near-Letter-Quality Printing

If you do not have a near-letter-quality dot matrix printer you can consider printing your text files in either emphasized or doublestrike mode to obtain bolder looking text. As indicated in Table 6.2, ESC E and ESC F select and cancel emphasized printing, whereas ESC G and ESC H select and cancel doublestrike printing.

Because each of the methods required to enable and disable doublestrike and emphasized printing requires the use of an escape character, you can use `<ALT> 155` to enter the required escape code into a batch file. Then, you can follow the escape code by the uppercase character that represents the second part of the code sequence required to place the printer into a desired print mode. Thus, you could enter the sequence `<ALT> 155 G`, which when echoed to the printer would turn on doublestrike printing.

Program Listing 6.7 illustrates the creation of a batch file named DBLSTK.BAT, which switches the printer to doublestrike mode, prints a desired text file, and then cancels the doublestrike mode, thereby returning the printer to its normal print mode. You will note that DBLSTK.BAT is very similar to the COMPRESS.BAT program previously discussed; the only differences between them are in the printer control codes used in each program.

Program Listing 6.7

DBLSTK.BAT Program

```
ECHO <ALT>155G > LPT1:
TYPE %1 >LPT1:
ECHO <ALT>155H > LPT1:
```

Likewise, you can print the SAMPLE.TXT file in doublestrike mode by entering the command

```
DBLSTK SAMPLE.TXT
```

Rather than create a third separate batch file to perform emphasized printing, let us now create one program that incorporates several print modes. Then we can use one replaceable parameter to select the desired print mode and a second replaceable parameter to select the file to be printed.

Program Listing 6.8 lists a program named PRTCHNG that can select one of three print modes, print a file in that print mode, and then return the printer to its normal print mode.

Program Listing 6.8

PRTCHNG.BAT File

```
ECHO OFF
IF %1==COMPRESS GOTO :COMPRESS
IF %1==DBLSTK GOTO :DBLSTK
IF %1==EMPHS GOTO :EMPHS
ECHO WRONG PRINT MODE SELECTED
GOTO :END
```

```
:COMPRESS
ECHO <ALT>15 > LPT1:
TYPE %2 > LPT1:
ECHO <ALT>18 > LPT1:
GOTO :END
:DBLSTK
ECHO <ALT>155G > LPT1:
TYPE %2 > LPT1:
ECHO <ALT>155H > LPT1:
GOTO :END
:EMPHS
ECHO <ALT>155I > LPT1:
TYPE %2 > LPT1:
ECHO <ALT>155J > LPT1:
:END
```

The format of the command line entry to execute the PRTCHNG.BAT program is

```
PRTCHNG printmode filename
```

Here the print mode must be either COMPRESS, DBLSTK, or EMPHS, which together define three named routines in the program that initiate compressed, doublestrike, and emphasized printing, respectively.

After line 1 turns the echoing of commands off, the three IF statements check the value of the first replaceable parameter entered in the command line. If it equals COMPRESS, DBLSTK, or EMPHS, a branch to the appropriate location in the program occurs to perform the indicated printing operation. If no match occurs, the program displays an error message, branches to the label END, and terminates. In this program the %2 replaceable parameter specifies the file to be printed. Although the program does not check to ensure that a valid file exists, you could add an IF EXIST batch command to test for the file's existence. Then, if the file does not exist, the program would display an appropriate error message and branch to the label END.

You can easily modify the file listed in Program Listing 6.8 to add other printer modes. Use of the ALT key and the technique described for entering an escape (ESC) character enable you to develop other printer control routines that are similar to the ones given here.

PRINTER PORT TESTING

Because parallel printer ports are included on video adapters, multifunction boards, and in some instances even on disk controllers, it is quite common for many computer systems to have two or more parallel ports. This situation can cause several problems. First, if two or more parallel ports are set to the same address using configuration switch settings, an address conflict will eventually occur when you attempt to send data to your printer. If two or more ports are assigned to the same printer address (LPT1, LPT2, or LPT3) they can cause the operating system to become confused and freeze the whole computer system. Normally the only way to unfreeze your computer is to turn off the power to your system unit, which also destroys all work that you have not saved to disk.

A second problem can arise when you are installing a new system. If, upon examining the rear of the system unit, you find several unlabeled 25-pin connectors that can represent the output of parallel printer ports, serial ports, or even IBM's Synchronous Data Link Control (SDLC) adapter, you have no way of telling which of the 25-pin connectors are the parallel printer ports, and to which particular port you connect your parallel printer.

In this section we will examine the use of a simple batch file that can be used to determine if there is an addressing conflict as well as to connect your printer to an appropriate parallel printer port.

Printer Testing

Program Listing 6.9 illustrates the creation of a one-line batch file that you can use to test for a parallel printer port address conflict, as well as to determine if you have cabled your printer to the correct parallel printer port. To execute this program, enter the command PTRTST followed by the printer address you wish to test—LPT1, LPT2, or LPT3.

Program Listing 6.9

Parallel Printer Test Program

```
ECHO THIS IS A TEST OF %1 > %1
```

By entering the command

```
PTRTST LPT1
```

the address LPT1 is assigned to the replaceable parameter %1, so the message "THIS IS A TEST OF LPT1" is routed to the device whose address is LPT1. Similarly, entering LPT2 after the command displays the message "THIS IS A TEST OF LPT2" on the device whose address is LPT2, and so on.

A few words of caution concerning the use of the PTRTST batch file are in order.

First, if two ports are assigned to the same address, your computer will freeze, because the use of the program will only verify an addressing conflict. If this situation occurs, you will have to turn off power to your system unit, disconnect your power cables, and open the cover to your system unit. Then, using the manuals that define the configuration switch settings on adapter boards that include parallel printer ports, you must determine the address assigned to each parallel printer port and change one or more addresses to remove the addressing conflict. As an example, if your video monitor adapter and a multifunction board both had parallel ports that were assigned device address LPT1, you would have to change the address of one port to device address LPT2.

A second problem that can occur with the use of the PTRTST batch file concerns determining the appropriate printer port to connect to your printer. You will probably prefer to cable your printer to the parallel printer port assigned device address LPT1. This is because most application programs use that address as the default for sending data to the printer. If you have two or more 25-pin connectors at the rear of your system unit and do not know which one is a printer port, the execution of the PTRTST batch file may also freeze your computer system. This can occur, as an example, if you cable your printer to a 25-pin connector that represents a serial port and enter the command line PTRTST LPT1. Doing this causes the batch file to attempt to send the test message to the printer port; however, because no printer is powered on and cabled to that port, a device fault will occur. This device fault is the method by which the operating system denotes that it did not receive a control signal from the printer. This circumstance occurs when the printer is cabled to the wrong port. Because the batch file does not service the device fault, your system will either display an error message or freeze, depending upon the version of the operating system you are using. If this should happen, either perform a system reset by pressing the CTRL-ALT-DEL key combination, or power the system unit off and then back on. Then, by cabling the printer to the next port you can continue your testing using the PTRTST batch file.

Printer Port Testing

7

Utility Routines

As its name implies, this chapter's focus is utility routines. We'll create a total of 16 utility routines in this chapter, ranging in scope from DOS command help files to routines that enable text file searches and database operations.

The techniques presented in this chapter, like those presented in Chapters 4 through 6, can either be used as self-contained entities or merged with other routines you may wish to construct to form a single program that will meet your specific requirements. While a few of the routines in this chapter may at first glance appear to be simple, you should note that each section is also designed to illustrate one or more DOS concepts. The goals of this chapter are to give you a firm understanding of the operation of DOS commands and to show you specific techniques that can boost the productivity of your computer system.

CREATING DOS COMMAND HELP FILES

If you or your users operate an IBM PC or compatible computer and are not well versed in the use of DOS commands, online assistance can be a most helpful feature to provide. In this section we will create a batch file named HELP.BAT that enables users to enter the command HELP followed by a DOS command to obtain online assistance concerning the use of the specific command.

Program Listing 7.1 illustrates the creation of the HELP.BAT file.

Program Listing 7.1 **Creating the HELP.BAT File**

```
COPY CON: HELP.BAT
ECHO OFF
IF EXIST %1.HLP GOTO LOCATE
IF %1==0 GOTO HELP
ECHO HELP NOT AVAILABLE FOR %1
```

```
GOTO END
:LOCATE
TYPE %1.HLP
GOTO END
:HELP
TYPE HELP.HLP
:END
```

After turning ECHO off, the IF command checks for the existence of a file whose name is the first replaceable parameter used when you run the program and whose extension is .HLP. If the file exists, a branch to the label LOCATE occurs, after which the TYPE command displays the file whose name was entered in the command line when the batch file was initiated and whose extension is .HLP.

If the name of the DOS command assigned to the replaceable parameter %1 does not exist with the extension .HLP in the current directory, the second IF command executes. This IF command is included in the batch file to avoid the need to type HELP HELP in order to display a master HELP file that explains how to obtain help for selected DOS commands. Instead, entering HELP by itself assigns the replaceable parameter %1 a null value of "". Because this value creates a match in the string comparison, a branch to the label HELP occurs. Then, the next line in the batch file displays the HELP.HLP file on the screen. If you plan to have more than one screenful of help on a topic, you should incorporate the MORE command so that the text doesn't scroll off the screen before it's read.

If you create separate ASCII files that explain the use of such DOS commands as COPY, DIR, FORMAT, ERASE, and so on, you only have to enter the command

```
HELP COMMAND
```

Creating DOS Command HELP Files

where COMMAND is a DOS command for which there is an ASCII file explaining the use of the command. Then, the contents of that file will be displayed to give on-line assistance.

Program Listing 7.2 shows the code for the program file ERASE.HLP, created here using the EDLIN text editor contained on the DOS diskette. The file contains text that explains the use of the ERASE command. Figure 7.1 shows exactly what you'll see on the

Figure 7.1

Obtaining Online Assistance from the ERASE Command

```
A>help erase

A>echo off
The ERASE command is used to erase a file from disk.

The format of ERASE is:
                         ERASE  [d:][path]filename[.ext]
     where:
            [d:]   - disk drive where file is located
            [path] - directory path to the file
            filename - name of the file to erase
            [.ext]   - filename extension (if present)

  Examples:
            ERASE B:\123\GRAPH1.PIC

            ERASE SAMPLE.HLP

A>
```

screen if you enter HELP ERASE to obtain information on the use of the DOS ERASE command.

Program Listing 7.2

Creating ERASE.HLP

```
 1:*The ERASE command is used to erase a file from disk.
 2:*
 3:*The format of ERASE is:
 4:*                          ERASE  [d:][path]filename[.ext]
 5:*    where:
 6:*              [d:]   - disk drive where file is located
 7:*              [path] - directory path to the file
 8:*              filename - name of the file to erase
 9:*              [.ext]   - filename extension (if present)
10:*
11:*  Examples:
12:*              ERASE B:\123\GRAPH1.PIC
13:*
14:*              ERASE SAMPLE.HLP
```

Expansion to Application Programs

You can also create other ASCII text files with the extension .HLP that provide online assistance for application programs. As an example, assume you have two application programs named BEST and WORST for which you would like to provide users with online assistance. You could simply create text files with EDLIN named BEST.HLP and WORST.HLP, in the same way in which you created the ERASE.HLP file (Program Listing 7.2). Then, entering the command HELP BEST (or HELP WORST) would result in the HELP.BAT file displaying the contents of BEST.HLP (or WORST.HLP) on the screen.

On-line assistance can be useless if computer users do not know how to access required information. To assist novices who are the best candidates for using help files, consider modifying the master menus presented in Chapter 4 in the section Creating File Reference Commands. As an example, at the bottom of a master menu you could display the three-line message

```
FOR ASSISTANCE IN USING DOS COMMANDS ENTER
HELP XXXXX
WHERE XXXXX CAN BE ONE OF FOLLOWING COMMANDS
COPY DIR ERASE
```

As an alternative to displaying a three-line message at the bottom of a master menu, you can consider creating a file named COMMAND.HLP. This file could contain a description of how to use your online HELP facility, as well as the commands and/or programs for which assistance is available. Then, you could either simply post a note that says "type HELP COMMAND for assistance," include the command on your application's quick reference card, or perhaps include information in your organization or client's newsletter concerning entering this simple command to obtain knowledge of the online assistance facility.

A DOS DATABASE

The DOS filter commands give you the ability to perform a limited degree of database operations on ASCII files. For simple applications, DOS filter commands may eliminate the need to purchase a database management system. In other applications, easy-to-use DOS filter commands may be able to retrieve the required information faster than a database program.

In this section we will develop an application to illustrate database management techniques that use only DOS facilities. This ap-

Figure 7.2

Telephone Directory Record Format

Last Name	First Name	Initial	Area Code	Telephone Number	Department Code

plication shows how we can maintain the telephone directory of an organization and even extract departmental directory information.

Telephone Directory

Before illustrating the use of DOS commands to perform database operations, let us see how to create a database using the EDLIN text editor. You can also use a word processor that can create ASCII text files or the DOS COPY command to copy data directly from the keyboard to a file.

Assume that Figure 7.2 illustrates the record format of the telephone directory we wish to create. Also assume that we will permit up to 12 characters for the last name, 13 characters for the first name, 1 character for the person's initial, 3 characters for the area code, 9 characters for the telephone number, and 3 characters for the department code. Thus, each record will be 40 characters in length.

Program Listing 7.3 illustrates the use of EDLIN to create a file consisting of five records that illustrate the database manipulation capability of DOS commands.

Program Listing 7.3

Using EDLIN to Create a Sample Database

```
1:*MELVIN      FRED         X2015551212A06
2:*JONES       JONATHAN      3055451412B10
3:*NACHES      PAUL          4158201495A06
```

```
4:*PORTER      MARYANNE     Q3055451416B10
5:*HOOVER      HERBERT      H2015551218A06
```

Note that the records entered into the file were not typed in alphabetical order. While it might be easy to sort entries before creating the initial database, you do not have to do so, because the SORT command can be used to perform this activity. Even if the database *were* initially sorted, if you use EDLIN or a word processor to maintain the directory it would be easier to add new employees to the end of the file. Then, you could use the SORT command to resequence the records.

An example of the use of the DOS SORT command appears in Figure 7.3. In this example the output of the sort operation is directed to the computer screen for illustrative purposes. Obviously, you could also direct the output of the SORT command to the printer or to a file.

In the preceding example, the SORT command was used without parameters, so the sort operation began in column 1. Suppose you require a directory listing arranged by department within your or-

Figure 7.3

Sorting the Telephone Directory

```
A>sort <phonedir.txt
HOOVER      HERBERT      H2015551218A06
JONES       JONATHAN      3055451412B10
MELVIN      FRED         X2015551212A06
NACHES      PAUL          4158201495A06
PORTER      MARYANNE     Q3055451416B10

A>
```

ganization. To accomplish this you could use the SORT command, beginning the sort in column 37, as illustrated here. Note that when you sort on the department code, names will not be alphabetized within a department.

```
A>sort /+37 <phonedir.txt
MELVIN    FRED       X2015551212A06
NACHES    PAUL        4158201495A06
HOOVER    HERBERT    H2015551218A06
JONES     JONATHAN    3055451412B10
PORTER    MARYANNE   Q3055451416B10

A>
```

Suppose you also require a list of personnel assigned to a specific department. You could use the FIND command to extract each record containing a string that matches the department code of interest. The following example illustrates the use of the FIND command to obtain a list of all persons in department A06 on the printer.

```
A>find "A06" phonedir.txt > LPT1:
---------- phonedir.txt
MELVIN    FRED       X2015551212A06
NACHES    PAUL        4158201495A06
HOOVER    HERBERT    H2015551218A06

A>
```

Information Retrieval

If your telephone directory is extensive, you may prefer to develop a method to retrieve information from the database. The following one-line batch file named LOCATE.BAT retrieves records from the previously created PHONEDIR.TXT file.

```
A>copy con: locate.bat
Find "%1" phonedir.txt
^Z
        1 File(s) copied
```

The LOCATE.BAT file executes the DOS FIND command, with one replaceable parameter, in the previously created telephone directory file named PHONEDIR.TXT.

Once you've created the batch file, you can enter the command LOCATE followed by a string that will be used as a match for the records in the file. This string can be a name, department code, telephone number, area code, or other "field" that is assigned to the replaceable parameter %1. Figure 7.4 shows the result of executing

Figure 7.4

Locating Records with a 305 Area Code

```
A>locate 305

A>find "305" phonedir.txt

---------- phonedir.txt
JONES       JONATHAN      3055451412B10
PORTER      MARYANNE     Q3055451416B10

A>
A>
```

UTILITY ROUTINES

the LOCATE.BAT batch program with an area code of 305, which will be assigned to the replaceable parameter.

Modifications to Consider

With EDLIN and the DOS FIND and SORT commands you obtain the ability to perform many database operations that, in some instances, would at first glance appear to be satisfied only by the use of a database program. Because the FIND command can extract all records containing or not containing a predefined string, this command also enables you to create DOS programs to update your database. As an example, suppose you desire a mechanism to remove entries from the telephone directory. You could use EDLIN to remove line entries; alternatively, Program Listing 7.4 shows you how to create a command named REMOVE that can accomplish the same function.

Program Listing 7.4

Program to Remove Entries

```
COPY CON: REMOVE.BAT
FIND "%1" /V PHONEDIR.TXT >TEMP.$$$
COPY TEMP.$$$ PHONEDIR.TXT
ERASE TEMP.$$$
```

In this example, the FIND command locates all records in the file PHONEDIR.TXT that do *not* contain the string assigned to the replaceable parameter %1, and sends those records to a temporary file named TEMP.$$$. Next, the COPY command copies the contents of the temporary file to PHONEDIR.TXT, in effect updating the file to reflect the removal operation. Finally, the ERASE command deletes the temporary file.

AUDIO AND GRAPHICS GENERATION

You can generate an audio sound or display different graphics from DOS by using the ALT key and the numeric keypad. In this section we will first examine the concept behind creating batch files to generate an audio sound or different types of graphics. Next, we will create several batch files to illustrate the concepts and then examine how the concepts presented in this section can be integrated with other techniques.

The ASCII Code

The IBM PC and compatible personal computers use an extended 8-bit ASCII character code that defines 256 distinct characters. The first 32 characters, whose values are 0 through 31, are usually employed to control communications and are commonly known as *control characters.* Table 7.1 lists the meaning of these control characters in order of their values.

Although ASCII control characters are normally supposed to be nonprintable, IBM associated many graphic symbols with control characters. As an example, a smiley face is assigned to the ASCII character whose value is 002 while the heart, diamond, spade, and club symbols are associated with the control characters whose ASCII values fall in the range between 004 and 107.

To enter an ASCII character into a file, you can hold down the ALT key and enter the numeric value of the character from the numeric keypad. The only exception to this is the entry of the escape (ESC key) character whose decimal value is 027. As previously explained, the escape character is interpreted by DOS literally as a request to terminate the current activity. Thus, you must enter a disguised escape character by adding 128 to 027, for a decimal value of 155. This value will be interpreted by most printers as an escape

Table 7.1

ASCII Control Characters

ASCII Value	Mnemonic	Meaning
00	NUL	Null character
01	SOH	Start of Header
02	STX	Start of Text
03	ETX	End of Text
04	EOT	End of Transmission
05	ENQ	Enquiry
06	ACK	Acknowledgment
07	BEL	Bell
08	BS	Backspace
09	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledgment
22	SYN	Synchronous Character
23	ETB	End of Transmission Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	Form Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

character, although it will not cause DOS to perform an escape command function when you enter the value from the keyboard.

Audio Alarm Generation

As you examine Table 7.1, note that the BEL control character can be used to beep the computer's speaker. Thus, if you can make DOS generate the control character whose ASCII value is 7, you will obtain a mechanism to generate an audio alarm.

Program Listing 7.5 contains a one-line batch file named BEEP.BAT whose execution causes the computer's speaker to beep. The ^G denotes CTRL-G, which is displayed when you press the ALT key and enter number 7 from the numeric keypad.

Program Listing 7.5

Creating a Speaker Beeper

```
C>COPY CON: BEEP.BAT
ECHO ^G
^Z

               1 File(s) copied
```

Once you've saved the batch file by entering a CTRL-Z (^Z), you can then simply enter the command BEEP to cause the computer's speaker to beep. Similarly, you can either include an ECHO statement containing an ALT-007 key combination or you can have any batch file execute a BEEP batch file when you wish to make an audible point.

Graphics Generation

Although DOS cannot be used to generate picture element (pixel) graphics, it does produce graphics images based upon the display of different ASCII characters. Many ASCII characters whose values are above 127 generate block graphic characters, such as double horizontal or vertical bars and upper and lower right and left corners. Most BASIC manuals contain an appendix that lists all ASCII codes and their resulting display. As an example of the use of block graphics, review the ASCII codes described in Table 7.2.

Suppose you want to create a box using the block graphic characters listed in Table 7.2. You can press the ALT key and enter the three numeric digits that represent the required block characters. Program Listing 7.6 illustrates the creation of a file named BOX.BAT that produces a box image. In creating this file the use of the ALT key was placed by the author within greater than and less than symbols to indicate that it must be pressed while the following numeric value is entered from the numeric keyboard. Note that the character whose value is 032 is the space character, which produces a blank space on the computer screen.

Program Listing 7.6

Creating a Box Image

```
COPY CON: BOX.BAT
<ALT>201<ALT>205<ALT>187
```

Table 7.2

Representative Block Graphic Characters

ASCII Value	Display
186	Vertical double bar
187	Upper right corner
188	Lower right corner
200	Lower left corner
201	Upper left corner
205	Horizontal double bar

```
<ALT>186<ALT>032<ALT>186
<ALT>186<ALT>032<ALT>186
<ALT>200<ALT>205<ALT>188
^Z
```

By entering the command TYPE BOX you can display a box constructed from ASCII graphic images on the display. Similarly, entering the command TYPE BOX>LPT1: sends the box to your printer.

Through the use of ASCII graphic symbols, you can call attention to specific areas on the screen by enclosing text in boxes or rectangles. This can be accomplished by mixing combinations of the ALT and numeric keys with text to meet your specific requirements. Then, you can either incorporate the resulting routine into an existing batch file or use it as a separate batch file to perform these specific functions.

DISK DIRECTORY LABEL GENERATOR

Although most professionally produced software programs have a label on their floppy diskettes, such labels merely list the program's name and perhaps a copyright notice. While you normally use a write protect tab to protect the contents of original vendor-sold diskettes, it is often quite useful to have a listing of the files on your duplicate working copy diskettes. In addition, as you create correspondence using a word processor or models using a spreadsheet, it is helpful to have a record of the files on the diskettes in your library.

In this section we will create a batch file that can be used to print labels containing the contents of a specific diskette. Many users may prefer simply to create a printed list of the files on a diskette, cut the listing to a square, and place that list inside the diskette envelope, so the batch file will generate labels that can fit into 5¼ inch wide sleeves.

Program Listing 7.7 contains the batch file named PRTLABEL.BAT that was created to print the contents of a disk in a compressed print mode. The prefix PRT was added to LABEL to avoid confusion with the DOS command LABEL that became available in DOS 3.0 and that is used to create a disk volume label of up to 11 characters.

Program Listing 7.7

PRTLABEL.BAT File

```
1:*echo off
2: cls
3: echo PRINTING LABEL FOR DISK....please wait...
4: dir %1 ¦find /V "Volume in drive" ¦find /V "bytes free" >TEMP.$$$
5: echo ^O >LPT1
6: type TEMP.$$$ >LPT1:
7: del TEMP.$$$
8: echo ^R >LPT1:
9: echo LABEL PRINTING COMPLETED...
```

Line 1 turns off the echo of the commands that follow. Line 2 clears the screen, and line 3 informs you when label printing has started. Line 4 uses the piping and filter capabilities of DOS for several actions. The first part of line 4 uses the DOS DIR (directory) command to get a directory of the disk specified. Because the default disk directory is the current drive, you must specify a different drive such as B by issuing the command PRTLABEL B:, where B: is assigned to the replaceable parameter %1. The output from the directory command (DIR %1) is passed to the next part of line 4 in the PRTLABEL batch file. This next part uses the DOS FIND command with the /V parameter to find every line in the directory that does not contain the string `"Volume in drive"`. The output is then passed to the next part of line 4, which uses FIND on every line in the directory except the lines containing the string `"bytes free"`.

These two FIND sequences essentially remove unwanted lines from the directory listing, thereby filtering out information unnecessary for the disk label. The final part of line 4 passes the resulting information to a temporary file on disk named TEMP.$$$. This file is actually created on disk, but is erased later in this batch file.

Line 5 sends a special control code to the current printer device—LPT1 in our example. Notice that this control code is created by pressing the CTRL key and the uppercase O key at the same time, which is displayed as ˆO. This symbol can also be created by using the ALT key and the numeric keypad on the keyboard. In either case, the ECHO command sends the code that turns on compressed printing on an Epson or Epson-compatible printer.

Because printers from different vendors may use different codes, be sure to enter the correct code for the particular printer you are using. The second method of entering this code is to press and hold the ALT key, then type the appropriate ASCII decimal value on the numeric keypad. Because this example was done on an Epson RX-80 printer, the control code was entered by pressing and holding the ALT key, typing the number 15 on the numeric keypad, and then releasing the ALT key. This key sequence is the same as a ˆO, even though it was entered as an ALT 15.

In line 6 of the program, the DOS command TYPE sends the file created previously (TEMP.$$$) to the current printer. This effectively places the directory, without unwanted information, on the printer in a compressed print mode. The compressed mode fits more information into a smaller area, therefore fitting more files on a disk label to fit inside the diskette jacket. If you have a large number of files on a disk, even compressed mode may generate a listing that is more than 5¼ inches long. In such instances, you should use the /W switch to produce a wide listing.

Line 7 deletes the TEMP.$$$ file created earlier, whereas line 8 turns compressed mode off at the printer. The ASCII code used in line eight is CTRL-R (ˆR) or ALT 18 for the Epson printer.

The control codes used in the program are ANSI control characters. Because of this you must load the file ANSI.SYS prior to using the PRTLABEL.BAT program. This procedure is normally accomplished by placing the DEVICE=ANSI.SYS statement in the CONFIG.SYS file on your system as explained earlier in the discussion of printer mode control techniques.

Line 9 announces when label printing is complete. A sample of the output generated on the printer is shown in Figure 7.5, where the program was executed by entering the command `PRTLABEL B:` to obtain the directory on drive B.

As indicated, the use of the PRTLABEL program provides an effective mechanism for printing disk labels. These labels can be easily inserted into diskette sleeves, or you could use the program to obtain disk listings to place into a reference book. For either situation, the

Figure 7.5

Directory Listing Printed in Compressed Mode

```
Directory of B:\

SAMPLE   XYZ       206  11-08-87   2:53p
PRTLABEL BAK       242  11-18-87   9:29p
PRTLABEL BAT       228  11-18-87   9:50p
123          <DIR>      11-08-87  11:59a
DBASE        <DIR>      11-08-87  11:59a
```

When the program is executing the following message is displayed on the screen:

```
PRINTING LABEL FOR DISK....please wait...
LABEL PRINTING COMPLETED...
```

program provides an easy mechanism to observe the contents of your disks quickly.

ELIMINATING ILLEGAL SUBDIRECTORY NAMES AND FILENAMES

Although the DOS command processor (COMMAND.COM) file does not accept spaces in directory names or filenames, some application programs do. Such application programs bypass the use of COMMAND.COM, passing directory and filenames as an ASCII string to a file control block in the operating system. When this occurs, you can include spaces as well as other special characters in the filename without causing an error.

Although you can continue to use the application program to manipulate previously created directories and files, these directories can represent a problem if you use other programs or attempt to remove them through DOS. To remove spaces or other illegal characters in filenames, you must trick COMMAND.COM by using global characters. As an example, assuming you previously created a file with the name GIL HELD, you could rename that file using the DOS RENAME command:

```
RENAME GIL?HELD GILHELD
```

Because the wildcard character (?) matches the space or any other illegal character, DOS will find the file and rename it. Unfortunately, the RENAME command does not work with subdirectories; a different technique is therefore required to handle illegal characters in subdirectory names.

Although subdirectories cannot be renamed, they can be removed. Instead of using the DOS RD command, you must use the equivalent BASICA command, because the latter permits you to define a directory name as a string. Using the DOS RD command on a directory name that contains a space would cause an error, because the name would not appear to COMMAND.COM as a single entity.

To remove a directory, you must first remove all files and subdirectories under that directory. After that is accomplished, you must relocate your current directory to the parent of the directory to be removed. As an example, if the directory GIL HELD under the root directory was to be removed, you would enter CD\.

Then, you would execute BASICA or GW-BASIC to initiate the BASIC interpreter and enter the following BASIC command to remove the directory:

```
RMDIR "GIL HELD"
```

PROGRAMMING FUNCTION KEYS AND KEYBOARD MACROS

It is relatively simple to program function keys in BASIC, using the built-in facilities of that language. Unfortunately, there are no direct statements or commands in DOS that provide this programming capability; however, using the code values assigned to individual keys and key combinations as well as the DOS built-in American National Standards Institute (ANSI) terminal driver, you can avoid purchasing keyboard macro programs to handle this task.

Before creating a batch file to change function key meanings, you must ensure that the ANSI.SYS file is loaded whenever you power on the computer or reset the system. This is accomplished by adding the statement:

```
DEVICE=ANSI.SYS
```

to the CONFIG.SYS file on your computer system. Next, you must understand the code sequence required to reconfigure the key or key combination to which you desire to assign a new value. The format to assign a new value to a key or key combination is

```
ESC[initial keycode]; new keycode p
```

In the preceding format, a lowercase *p* must be used to terminate the remapping command.

Using Function Keys

The initial keycode is an ASCII decimal that represents the value of the key you wish to reassign. Table 7.3 lists the initial keycodes of the function keys on an IBM PC or compatible.

Table 7.3

Initial PC Function Key Keycodes

Function Key	**Initial Keycode**
F1	0; 59
F2	0; 60
F3	0; 61
F4	0; 62
F5	0; 63
F6	0; 64
F7	0; 65
F8	0; 66
F9	0; 67
F10	0; 68

When you use the function keys, it is crucial to know the values of the extended codes that define the keys. Extended codes are used to represent certain keys and key combinations that cannot be represented in standard ASCII code.

To distinguish an extended code from standard ASCII code, a null character is used as a prefix for a two-character string. The microprocessor of the computer examines the ASCII code of characters sent from the keyboard. If an ASCII code of 0 is encountered, this signifies that the following code is the second part of a two-character string that defines an extended code. Thus, the first zero for each function key is an ASCII null character, which indicates that the second code is actually the second character in a two-character extended code. Due to this, the initial keycode for each function key consists of the null character separated from a second code by a semicolon.

The new keycode can be an ASCII decimal, a string enclosed in double quotes, or a combination of ASCII decimal characters and strings separated from one another by semicolons. If you simply use an ASCII decimal, you in effect remap the function key to another key. It would be more desirable to remap the keys to execute predefined DOS commands, so you could increase your productivity by developing a batch file that issues DOS commands when users press function keys. For example, to reprogram the F1 key to issue the DOS command CD \LOTUS followed by a carriage return, the new keycode `"CD \LOTUS" 13;p` must be issued. Here the 13 is the value of a carriage return in ASCII. Thus, the first line in your batch file to remap the F1 key to issue the command CD\LOTUS would be:

```
ESC[0;59;"CD \LOTUS";13p
```

where ESC represents the escape character.

Program Listing 7.8 presents a batch file named KEY.BAT whose execution will assign DOS commands to the first 10 function keys. Using this technique, you can develop a similar batch file to remap as many function keys as you need to specific DOS commands to meet your requirements.

Program Listing 7.8

KEY.BAT Program to Assign Strings to Function Keys

```
 1:*prompt $e[0;59;"CD \LOTUS";13p
 2: prompt $e[0;60;"CD \DBASE";13p
 3: prompt $e[0;61;"CD \TELCOM";13p
 4: prompt $e[0;62;"BASICA";13p
 5: prompt $e[0;63;"DIR /W";13p
 6: prompt $e[0;64;"TREE";13p
 7: prompt $e[0;65;"DIR A:";13p
 8: prompt $e[0;66;"DIR B:";13p
 9: prompt $e[0;67;"DIR C:";13p
10: prompt $e[0;68;"CLS";13p
11: prompt
```

In this listing, the escape character is generated by the DOS PROMPT command, which enters `PROMPT $e`. The e after the $ sign must be in lowercase. Note that line 1 assigns the command `CD \LOTUS` to the F1 function key. Now when a user simply presses F1 at the DOS prompt, the command `CD \LOTUS` is issued and executed. Line 2 remaps F2 to the command `CD \DBASE`. Line 3 remaps F3, and so on. Line 11 returns the PROMPT value to its normal sign. To make each of these key assignments effective, you need only issue the KEY command, which executes the KEY.BAT file. As an alternative, the KEY.BAT file could be executed from the AUTOEXEC.BAT file at the time the computer is powered on.

Creating Keyboard Macros

Using the previous example of remapping function keys to perform predefined tasks, it becomes apparent that any key on the keyboard could be remapped to a command or string of commands. Because most, if not all, of the letter keys are normally used for typing, it would be inappropriate to remap letters. Other available keys you can consider are multiple key combinations, such as the ALT key pressed simultaneously with an uppercase letter key or a combination of SHIFT, ALT, or CTRL pressed at the same time a function key is pressed.

Like the extended codes described for function keys, similar codes are defined for multiple key sequences using the ALT, CTRL, and SHIFT keys in conjunction with other keys. To assist you in creating keyboard macros using other key combinations, Table 7.4 lists the second code of the extended code values of many of the commonly used key combinations that commercial programs employ. As you will soon see, knowing these extended codes will enable us to create keyboard macros and assign such macros to a variety of key combinations.

Although the 68 key combinations listed in Table 7.4 should suffice for almost all your keyboard macro creations, other keys and key combinations represented by extended codes are available for use. Refer to an appendix in your BASIC or GW-BASIC manual for a list of all the extended codes supported by your computer.

Assume you want to list the directory of drive A, B, or C by simply pressing ALT-A, ALT-B, or ALT-C, respectively. The batch file you will examine here assigns the commands required to list the directory of each drive to the appropriate ALT key combination.

From Table 7.4 note that the second code of the extended codes for the ALT-A, ALT-B, and ALT-C key combinations is 30, 48, and

Table 7.4

Extended Codes of Popular Key Combinations

Key Combination	Second Code	Key Combination	Second Code	Key Combination	Second Code
ALT-Q	16	ALT-B	48	ALT-F1	104
ALT-W	17	ALT-N	49	ALT-F2	105
ALT-E	18	ALT-M	50	ALT-F3	106
ALT-R	19	Shift-F1	84	ALT-F4	107
ALT-T	20	Shift-F2	85	ALT-F5	108
ALT-Y	21	Shift-F3	86	ALT-F6	109
ALT-U	22	Shift-F4	87	ALT-F7	110
ALT-I	23	Shift-F5	88	ALT-F8	111
ALT-O	24	Shift-F6	89	ALT-F9	112
ALT-P	25	Shift-F7	90	ALT-F10	113
ALT-A	30	Shift-F8	91	ALT-1	120
ALT-S	31	Shift-F9	92	ALT-2	121
ALT-D	32	Shift-F10	93	ALT-3	122
ALT-F	33	Ctrl-F1	94	ALT-4	123
ALT-G	34	Ctrl-F2	95	ALT-5	124
ALT-H	35	Ctrl-F3	96	ALT-6	125
ALT-J	36	Ctrl-F4	97	ALT-7	126
ALT-K	37	Ctrl-F5	98	ALT-8	127
ALT-L	38	Ctrl-F6	99	ALT-9	128
ALT-Z	44	Ctrl-F7	100	ALT-0	129
ALT-X	45	Ctrl-F8	101	ALT--	130
ALT-C	46	Ctrl-F9	102	ALT-=	131
ALT-V	47	Ctrl-F10	103		

46, respectively. Knowing this enables you to create the file KEYSETUP.BAT in Program Listing 7.9.

Program Listing 7.9

KEYSETUP.BAT Program

```
1:*echo off
2: prompt $e[0;30;"DIR A:";13p
3: prompt $e[0;48;"DIR B:";13p
4: prompt $e[0;46;"DIR C:";13p
5: echo keys ALT-A, ALT-B, and ALT-C now set for directories
6: prompt
```

In this listing line 1 turns the screen echo off, which will inhibit the display of the other commands in the file. Lines 2 through 4 use the DOS PROMPT command to reconfigure the ALT-A, ALT-B, and ALT-C key combinations. In each line the $e represents the escape character, and the first two numbers represent the extended code of each key combination. Next, the string in quotes represents the initial portion of the command assigned to each key combination. Because each command assigned to a key combination should immediately execute when the appropriate key combination is pressed, a carriage return must complete the assignment. You can do this by adding the ASCII code for the carriage return whose value is 13, as indicated in each line.

Line 5 simply provides a message that the three ALT key combinations are set for directories, whereas line 6 resets the PROMPT command.

The information presented in this section can be used either by itself or in conjunction with other techniques described in this book. For example, you may consider combining keyboard macros with master menus to provide users with the ability to navigate rapidly through hierarchical directory structures.

PERFORMING OPERATIONS AT DEFINED TIMES

One of the problems associated with developing a batch program to initiate an operation at a predefined time is the fact that the DOS IF command requires an identical match when string comparisons are used. To understand this problem, consider the following batch program segment.

```
:LOOP
IF TIME == 18:30 GOTO :NEXT
GOTO LOOP
:NEXT
```

Time on a PC is computed to an accuracy of a hundredth of a second, so it is very difficult for the comparison between the value retrieved from the DOS TIME command and the time the branch is to occur, 18:30 in this example, to be an exact match. Due to this there is an excellent probability that the preceding segment will result in a continuous loop.

To better visualize the problem associated with the use of the TIME command in a batch file, we will modify the previously created batch file segment. This modification will first set the TIME to 18:29:50.90, where .90 is 90 hundredths of a second. Next we will display the time each time the program goes through the loop. The modified program, named TIMELOOP.BAT, appears in Program Listing 7.10.

Program Listing 7.10

TIMELOOP.BAT Program

```
*echo off
 time 18:29:50.90
```

```
3: :loop
4: if time==18:30 goto :next
5: time <cr.bat
6: goto :loop
7: :next
8: echo out of loop...
```

In line 2 the time is set to 6:29 p.m., 50 seconds, and 90 hundredths of a second. Line 4 causes a branch to the label NEXT at precisely 6:30 p.m. In line 5 the DOS TIME command is used to display the current time; however, this command in actuality displays the current time and prompts the user to enter the new time. Although you could simply press the ENTER key when prompted to `Enter new time`, a simpler and more automated method is obtained by creating the file CR.BAT that contains the carriage return character. Then, the contents of that file can be input to the TIME command as indicated by the use of the less than sign in line 5.

The CR.BAT file can be created by entering the command:

```
COPY CON: CR.BAT
```

The only entry in this file is a carriage return that you enter by pressing and holding down the ALT key while you enter the decimal value 13 from the numeric keyboard. This is the ASCII value of the carriage return.

The TIMELOOP.BAT program is executed when you enter the command TIMELOOP. As indicated in the following sample, the first time line 5 is executed the time is 18:29:56.66, whereas the next time through the loop the time exceeds the value of 18:30 that is tested for in line 4. The program would therefore go into an infinite loop unless terminated by a CTRL-BREAK key sequence.

```
A>timeloop
A>echo off
Current time is 18:29:56.66
Enter new time:
Current time is 18:30:06.55
Enter new time:
Current time is 18:30:16.49
Enter new time:
```

To avoid the limitations of using identical string matches in the DOS IF command, you can construct a batch file to execute a BASIC program. Then you can use the better time comparison feature of the BASIC language to return you to the batch file at the time you select.

Here is how to create the batch file XTIME:

```
C>edlin xtime.bat
End of input file
*1
        1:*basica xtime.bas
        2: REM ...next batch command goes here...
*
```

By entering the command XTIME, the previously created batch file executes the BASIC program XTIME.BAS, shown here:

```
list
10 WHILE TIME$<"18:30:00"
20 WEND
30 SYSTEM
Ok
```

Lines 10 and 20 form a WHILE-WEND loop that continues executing as long as the string TIME$ is less than 18:30:00. Once the time passes that value, the loop is terminated and line 30 passes control back to DOS. Because DOS was executing a batch file, control of the computer is passed to the line subsequent to the one that executed the BASIC program XTIME.BAS. Although that line in the batch file contained a REM command, if it is replaced by the name of an executable program it will cause that program to execute at 18:30.

TEXT SEARCH

Although many word processing programs include a search and/or replace feature, its capability is normally limited to one string at a time. In this section we will develop a batch file that can be used to search ASCII text files to locate the occurrence of different combinations of strings based upon the boolean logic operators AND and OR.

As we develop the batch file, assume that the command line format used to invoke its execution will be

```
SEARCH Filename 'String1'[Operator 'String2']
```

The operator will be either AND or OR, while both the boolean operator and the second string ('String2') are optional entries denoted by brackets in the above format. Thus, to execute the batch file you would enter one of the following command line formats:

```
SEARCH Filename 'String1'
SEARCH Filename 'String1' OR 'String2'
SEARCH Filename 'String1' AND 'String2'
```

Program Listing 7.11 illustrates the listing of the batch file SEARCH.BAT that provides the ability to search a file for the occurrence of one or two strings, with the latter capability enhanced by the use of either the boolean AND or the boolean OR operator.

Program Listing 7.11

SEARCH.BAT Program Listing

```
*echo off
copy %1 temp1.$$$
if "%4" == "" goto :1STRING
if "%3" == "OR" goto :OR
if "%3" == "Or" goto :OR
if "%3" == "or" goto :OR
if "%3" == "AND" goto :AND
if "%3" == "And" goto :AND
if "%3" == "and" goto :AND
goto :1STRING
:OR
find %2 TEMP1.$$$ > TEMP2.$$$
find %4 TEMP1.$$$ > TEMP3.$$$
type temp2.$$$ ¦find /V "---------"
type temp3.$$$ ¦find /V "---------"
goto :CLEANUP
:AND
find %2 TEMP1.$$$ > TEMP2.$$$
find %4 TEMP2.$$$ > TEMP3.$$$
goto :DISPLAY
:1STRING
find %2 TEMP1.$$$ > TEMP3.$$$
:DISPLAY
type TEMP3.$$$ ¦find /V "---------"
:CLEANUP
if exist temp1.$$$ del temp1.$$$
if exist temp2.$$$ del temp2.$$$
if exist temp3.$$$ del temp3.$$$
```

After line 1 turns ECHO off, line 2 copies the file to be searched to a temporary file named TEMP1.$$$. The format of the SEARCH command allows up to four replaceable parameters to be used in the command, the fourth being used to define a second string. Thus, if the fourth parameter is set to a null value, there is only one string to be used by the program. In that case, line 3 causes a branch to the label :1STRING if the replaceable parameter %4 is set to a null value.

Lines 4 through 6 examine the third replaceable parameter for three different ways a user might enter an OR operator. If one was entered, the program branches to the label :OR in line 11. Similarly, lines 7 through 9 test the third replaceable parameter against three ways the user could enter the AND operator. Then, if an AND operator was entered, the program branches to the label :AND in line 17.

Line 10 assumes that if the user did not enter a version of AND or OR, only one string would be searched for; the program therefore branches to the label :1STRING.

Lines 12 through 16 perform text searches based upon the OR operator being specified. Line 12 uses the DOS FIND command, with the %2 replaceable parameter that has the value of the first string, to locate all lines in the file TEMP1.$$$ that contain the first string. These lines are then placed in a temporary file named TEMP2.$$$. Similarly, line 13 uses the FIND command to locate all lines in the TEMP1.$$$ file that contain the fourth replaceable parameter that contains the value of the second string entered in the SEARCH command line. Here each line in the file that contains the second string is placed in another temporary file, TEMP3.$$$.

The TYPE commands in lines 14 and 15 display the contents of both the TEMP2.$$$ and TEMP3.$$$ files. However, the FIND /V command in each line ensures that any line containing nine dashes is not displayed. The piping of the TYPE command output into the DOS FIND filter eliminates the display of directory heading information that could occur if a string matched all or a portion of the name of a file. Finally, line 16 causes a branch to the label :CLEANUP, where lines 26 through 28 delete any temporary files created by the program. Note that the output can be routed to the line printer by adding >LPT1: to lines 14 and 15.

In line 17 the label :AND represents the entry point for the routine that will extract lines that contain both strings specified in the SEARCH command line. In line 18 the DOS FIND command uses the second replaceable parameter to locate all lines in the file TEMP1.$$$ that contain the first string. These lines are then saved in the file TEMP2.$$$. Next, line 19 uses the FIND command with the replaceable parameter %4 to locate all lines in the TEMP2.$$$ file that contain the second string entered on the SEARCH command line. Thus, all lines saved in the file TEMP3.$$$ in line 19 represent lines that contain both the first string and the second string entered on the SEARCH command line. Next, line 20 causes a branch to the label :DISPLAY in line 23. Line 24 then uses the TYPE command to display the contents of the file TEMP3.$$$, with the exception of lines that contain nine dashes.

In line 21 the label :1STRING serves as the entry point in the program for performing the required search when the user entered only one string in the SEARCH command line. Here line 22 used the FIND command with the replaceable parameter %2, which contains the value of the first string to search the file TEMP1.$$$. Then, all lines containing the first string are saved in the file TEMP3.$$$ and displayed by the TYPE command in line 24 of the program.

Search Examples

To illustrate the use of the SEARCH.BAT program, let us first list the contents of an ASCII text file. The following sample shows the contents of the ASCII file PAY87.DAT, which we will use as input to the

SEARCH program. For simplicity, only five lines of data were entered into the file; however, the SEARCH program can operate on extensive ASCII files without modification.

```
A>type b:pay87.dat
MELVIN      FRED          X2015551212A06
JONES       JONATHAN       3055451412B10
NACHES      PAUL           4158201495A06
PORTER      MARYANNE      Q3055451416B10
HOOVER      HERBERT       H2015551218A06

A>
```

Suppose you want to extract all records in the PAY87.DAT file that contain the string "ANNE". The following example illustrates the entry of the SEARCH command and the resulting display when you attempt to extract all records containing that string. You should note by this example that the string search performed is not by position within a record. Therefore, any records that contain the string "ANNE" would be displayed regardless of their location in the record. While this text search routine may not be as valuable as a conventional database, which permits searching by specific fields or by character positions within a field, it does enable you to extract all records that meet a predefined criterion. Then, you could simply observe the resulting display to find the record or records of interest.

```
A>search b:pay87.dat "ANNE"

A>echo off
        1 File(s) copied

PORTER      MARYANNE      Q3055451416B10
A>
```

Now assume you want to extract all records that have either the string "ANNE" or the string "212A06" in a record in the PAY87.DAT file. The next example illustrates the SEARCH command used with two strings separated by the OR operator to obtain the desired information.

```
A>search b:pay87.dat "ANNE" OR "212A06"

A>echo off
        1 File(s) copied

PORTER      MARYANNE      Q3055451416B10

MELVIN      FRED          X2015551212A06
A>
```

As indicated, the text search program in this section can supplement the capabilities of a display program that lacks this capability, or it can be used for other information retrieval purposes. In addition, the program to search text can be incorporated into other programs to satisfy your specific requirements.

CREATING A CALCULATOR

Many people use calculators for a multitude of computations, ranging from balancing checkbooks to solving complicated scientific equations. The need for a calculator program on the PC has spawned several software pop-up products that contain built-in calculators for convenience. To avoid purchasing a separate product, you can develop a simple calculator program using DOS commands and BASIC; it will perform many mathematical operations.

Program Listing 7.12 illustrates the contents of a batch file named CALC.BAT. This batch file uses the piping and filtering capabilities of DOS to pass information to BASIC, which then performs the calculations and filters the desired answer back to the computer's screen.

Program Listing 7.12

The CALC.BAT Calculator Program

```
C>edlin calc.bat
End of input file
*l
       1:*echo off
       2: if "%1" == "" goto :ERROR
       3: echo ? CHR$(61); %1 %2 %3 %4 %5 %6 %7 %8 %9 : SYSTEM ¦ BASIC ¦ FIND "="
       4: GOTO :END
       5: :ERROR
       6: echo .
       7: echo You must tell me what to CALCULATE...please give me a formula.
       8: echo .
       9: echo ...something like     CALC 5 * 4  ...or... CALC 21+3*7
      10: echo .
      11: echo .
      12: :END
*
```

Line 1 turns the echo of commands to the screen off so the user does not see the commands as they are executed. Line 2 simply checks to see if the CALC command has something to calculate. If the CALC command has been entered without any parameters, control is passed to the label :ERROR in line 5. Next, lines 6 through 11 display an error message, informing the user that the CALC program must have something to calculate. Line 3 is the key to the capability of the CALC.BAT file. This line begins by piping the following into BASIC:

```
? CHR$(61); %1 %2 %3 %4 %5 %6 %7 %8 %9 : SYSTEM
```

The question mark (?) is BASIC's abbreviation for the PRINT command. The CHR$(61) function is the ASCII decimal code for the equal (=) sign. The semicolon after the CHR$(61) function tells BASIC to keep the cursor on the same line immediately after printing the equal sign. The variables %1, %2, %3, and so on, are batch file variations that are entered with the CALC command. For instance, if you entered

```
CALC 2 * 7
```

the variable %1 would have the value of 2, the variable %2 would have the value of * and the variable %3 would have the value of 7. Because up to nine variables can pass batch files and commands, you can pass a maximum of nine possible variables to BASIC with the CALC command. Remember, though, that a variable can be a string of characters as long as there are no spaces (blanks) within the string. In this instance, entering

```
CALC 2*7
```

would give the %1 variable a value of 2*7, because the string of characters is not separated by a space. This is a very important point to remember, because it means that you could pass a complicated formula to BASIC as one simple variable. Later in this section you will see an example of the power of the built-in calculator program.

Finally, Program Listing 7.12 includes the colon (:) and the word *SYSTEM*, which are both used in BASIC—the colon to start a new command and the SYSTEM command to exit BASIC and return control to the operating system. Thus, the first part of line 3 is piped (passed) into BASIC by the DOS piping symbol (¦). BASIC then performs the computations and calculations requested, and the result is piped (passed) out of BASIC by the DOS piping symbol and into the DOS FIND command. The FIND command simply finds and displays only the lines passed from BASIC that contain the equal sign, because the line with the equal sign contains the result of the calculation. Line 4 then transfers control to the end of the CALC.BAT file, and the results are printed on the screen.

The following four examples show use of the CALC command to perform computations. Note that because BASIC is used for calculations, all functions available to BASIC are available to the CALC.BAT calculator file—including SIN, COS, TAN, and even the ability to convert hexadecimal numbers to decimal and vice versa. The only warning here is: Never use more than nine variables (or eight spaces) in the calculation being typed. In the three examples where CALC was followed by an operand, you will note that the result precedes the command line. This occurs because CALC initiates the BASIC interpreter, which is instructed to PRINT the value of the operand. When this occurs, BASIC displays the results two positions above the DOS CALC command invocation.

```
C>calc
C>echo off
.
You must tell me what to CALCULATE...please give me a formula..
.
...something like    CALC 5 * 4  ...or... CALC 21+3*7
.
.

C>





= 7464
```

```
C>calc 24*311



= .9092975

C>CALC SIN(2)



= 10620

C>calc 36*295
```

Expanding the Calculator's Abilities

We now can use the calculator with many varied and different formulas, and you may wish to create additional batch files that use the CALC.BAT file to perform one or more specific calculations. As an example, assume you had money to invest and wanted to know which option would produce the most interest and profit on your investment. In this simple example, we will assume you have $1,000 to invest and have an option of investing at an interest rate of 10 percent for 10 years in a certificate of deposit that is compounded monthly or in a savings account that is compounded quarterly.

The formula for compound interest is

Pn = Po * (1 + (I / M)) ^ (M * N)
where: Po = the initial principal invested
I = the interest rate being paid
M = the compounding periods in a year
N = the number of years of the investment
Pn = the resulting principal and interest after the investment period is completed

Our comparison of the resulting principal and interest at the end of 10 years then becomes

```
1000 * ( 1 + ( 0.10 / 4 )) ^ ( 4 * 10 )
```

versus

```
1000 * ( 1 + ( 0.10 / 12 )) ^ ( 12 * 10 )
```

Because the preceding formulas compute the future value of an investment based upon a fixed interest rate, a fixed number of compounding periods per year and fixed term, they are also commonly called future value (FV) formulas. Which investment would be better and how much difference is there between quarterly and monthly compounding? We could perform the calculations in just a few seconds with a financial calculator, but with only a personal computer it becomes a simple matter to create a batch file to call the CALC.BAT calculator to compute the alternatives for us. Because there may be many alternatives with varying amounts of compound periods, interest rates, and years of investment, our formula should be able to compute all types of options.

Creating a Calculator

Program Listing 7.13 illustrates the listing of a sample batch file called FUTURE.BAT. This file requires an input of four parameters: the principal, the interest rate, the compounding periods, and the number of years of the investment. Lines 2 through 5 ensure that all four variables are entered with the FUTURE command. Line 6 calls the CALC.BAT calculator program previously created and passes it the formula for computing the resulting future value of your investment.

Program Listing 7.13

Future Value Computation Program

```
 1:*echo off
 2: if "%1" == "" goto :ERROR
 3: if "%2" == "" goto :ERROR
 4: if "%3" == "" goto :ERROR
 5: if "%4" == "" goto :ERROR
 6: CALC %1*(1+(%2/%3))`(%3*%4)
 7: GOTO :END
 8: :ERROR
 9: echo .
10: echo Please enter PRINCIPAL, INT. RATE, COMPOUNDING PERIODS, and No. OF YEARS...
11: echo .
12: echo ...something like  FUTURE   1000 .085 4 10
13: echo .
14: echo .
15: :END
```

Following are five examples of using the FUTURE command to determine the future value of an investment based upon the deposit of a principal amount at a given rate of interest, with a defined number of compounding periods over a specified number of years.

```
C`FUTURE

C`ECHO OFF
.
Please enter PRINCIPAL, INT. RATE, COMPOUNDING PERIODS, and No. OF YEARS..
.
```

```
...something like FUTURE 1000  .085  4  10
.
.
C>



= 2685.062

C>FUTURE 1000  .10  4  10


= 2707.029

C>FUTURE 1000  .10  12  10


= 2332.636

C>FUTURE 1000  .085  12  10



= 2318.906

C>FUTURE 1000  .085  4  10
```

In the first example, the command FUTURE was entered without any parameters, resulting in the display of the error messages contained in lines 9 through 14 of the batch file. In the second example, the future value of $1,000 deposited for 10 years at a 10 percent interest rate compounded quarterly was determined to be $2,685.06. When compounding was increased to monthly in the third example, the future value increased to $2,707.03. Similarly, the fourth and fifth executions of the FUTURE program computed the future value of $1,000 deposited at an 8.5 percent interest rate for 10 years when compounding occurs monthly and quarterly. Here you gain the power of a rudimentary spreadsheet, because you can alter variable information and test the results.

Based upon this example, it becomes possible for you to create combined DOS and BASIC programs that can greatly exceed the computational capabilities of pop-up utility programs. In addition, because BASIC is included on most DOS diskettes, this capability is usually without cost.

Other Financial Computations

The capability of using formulas and passing them to the CALC command for computations opens many doors to creating additional commands. Two additional and very useful financial commands you can use form batch files that compare taxable yields versus tax-free yields to see which alternative is most advantageous in specific cases.

Program Listing 7.14 contains a listing of a batch file called TAXABLE.BAT.

Program Listing 7.14

TAXABLE.BAT Program

```
*echo off
if "%1" == "" goto :ERROR
if "%2" == "" goto :ERROR
if "%3" == "" goto :ERROR
if "%4" == "" goto :ERROR
CALC %1/(1-(%2+%2*(%3+%4)))
GOTO :END
:ERROR
echo .
echo Please enter TAX FREE YIELD, FEDERAL, STATE, and LOCAL tax rates ...
echo .
echo ...something like  TAXABLE .0875 .33 .28 .06
echo .
echo .
END
```

When you enter a tax-free yield, federal tax rate, state tax rate, and local tax rate to this program, it computes the comparable yield for a taxable investment. The formula used is:

$$Tn = To / (1 - (F + F * (S + L)))$$

where: To = the tax-free yield being compared against
F = the federal tax rate as a percentage
S = the state tax rate as a percentage
L = the local tax rate as a percentage
Tn = the taxable yield which would be required to equal the return of this tax-free yield

Note that this formula is in line 6 of the TAXABLE.BAT batch file and is passed to the previously created CALC command. The rest of the TAXABLE.BAT file is similar to the FUTURE.BAT file and is self-explanatory. Two examples of the execution of this batch file are shown here. The first example simply illustrates the error message displayed if the command is invoked without four parameters. The second example returns a value of 15.68 percent, which is the equivalent taxable yield you would have to earn on a taxable investment to match the return of a tax-free security yielding 8.75 percent if your federal, state, and local tax rates were 33 percent, 28 percent, and 6 percent, respectively.

```
C>taxable

C>echo off

.

Please enter TAX FREE YIELD, FEDERAL, STATE, and LOCAL tax rates...

.

...something like  TAXABLE .0875 .33 .28 .06

.
```

```
.

C>



= .1568663

C>taxable .0875 .33 .28 .06
```

As an alternative to the preceding calculation, you could create a command called TAXNOT.BAT to compute the tax-free yield comparable to any given taxable yield. This batch file is shown in Program Listing 7.15.

Program Listing 7.15

Program to Compute Tax-Free Yield Based Upon Taxable Yield

```
 1:*echo off
 2: if "%1" == "" goto :ERROR
 3: if "%2" == "" goto :ERROR
 4: if "%3" == "" goto :ERROR
 5: if "%4" == "" goto :ERROR
 6: CALC %1*(1-(%2+%2*(%3+%4)))
 7: GOTO :END
 8: :ERROR
 9: echo .
10: echo Please enter TAXABLE YIELD, FEDERAL, STATE, and LOCAL tax rates...
11: echo .
12: echo ...something like  TAXNOT .15 .33 .28 .06
13: echo .
14: echo .
15: :END
```

Note that the formula is only slightly different from the one for computing corresponding taxable yields and that the formula in line

6 is again passed to the CALC command for computations and calculations. Using this program enables you to enter the yield on a taxable security followed by your federal, state, and local tax rates to obtain the equivalent tax-free yield. Thus, between the TAXABLE and TAXNOT programs, you can easily compare yields of taxable and nontaxable securities, one of many calculations not available with conventional utility programs.

COMPUTER USAGE LOGGING

Due to the varied uses of personal computers and the number of people using them, many businesses and families find that one computer must be shared among several people. Thus, it is quite common for a business to possess several personal computers for use by employees, with the usage of computers shared among employees. Given this situation, it is very difficult for management to evaluate how much use each computer is actually getting and what software is used most often by employees. A manager armed with this information could quickly assess the need for additional software or hardware and could evaluate growing trends in computer usage in his or her organization to plan for future growth and expenditures.

To provide management with personal computer usage information, a need exists for a method of logging who is using the personal computers and what software they are using most often. Whereas most people do not enjoy being "watched over the shoulder," a simple method will be developed in this section for logging who is using the computer and what software is popular with these users. No method is foolproof against a determined and intelligent user, but the methods discussed here will be simple, yet effective, for most situations.

The Signon Program

A simple BASIC program can be used to keep track of who is using a computer. This program can be executed automatically each time the computer is turned on by placing the following line in the AUTOEXEC.BAT file of the boot disk (we are assuming most businesses have hard disk systems and boot directly from the hard drive):

```
BASIC SIGNON.BAS
```

This line first calls the BASIC interpreter to execute a program called SIGNON.BAS. The BASIC program will then ask for your name. The program writes a record to a file called USERS.DAT that contains your name and the current date and time. When the BASIC program is finished, the BASIC interpreter returns control to the operating system, and the next command line in the AUTOEXEC.BAT file is executed.

Note that if you are using a different BASIC—such as GW-BASIC that is furnished on the DOS diskette of many IBM PC compatible computers—the BASIC command would be replaced by the command GW-BASIC. In addition, if special BASIC commands are executed in the SIGNON.BAS program that require advanced BASIC, the BASICA command should be used.

Program Listing 7.16 shows lines 10 through 100 of the BASIC SIGNON program. These lines are used for comments about the program and what it is designed to do. The single apostrophe (') is a simple method in BASIC for entering the REM (remark) command. Everything on the line after the single apostrophe is considered a remark or comment and is ignored by the BASIC interpreter.

Program Listing 7.16

Initial Portion of SIGNON Program

```
10 '***************************************************************************
20 '**    SIGNON PROGRAM FOR USERS........as of 2-2-88
30 '**
```

```
40 '**  The purpose of this program is to maintain a log of all users who
50 '**  access and use this IBM PC. The user is prompted to enter a name,
60 '**  and a record is written to the file USERS.DAT containing the name
70 '**  (the first 30 characters only), the date, and the time of signon.
80 '**
90 '**  ...LAST CHANGE...2/3/88...Tom Dolittle
100'************************************************************************************
```

Next, lines 119 through 200 place a welcome message on the screen and ask you to log on (enter your name). The CLS command at line 119 effectively clears the screen so the WELCOME message is printed attractively. The PRINT commands in lines 120 through 190 put the WELCOME message on the screen. Notice line 155 uses the SOUND command to play a little siren sound to get your attention. This line has been added to simply "dress up" this program and is completely unnecessary for proper execution. Line 195 is used to set the string variable N$ to nulls (not blanks, but no character value at all), and line 200 uses the INPUT command to prompt you to enter your name, which is placed in the N$ string variable.

```
119 CLS
120 PRINT"*********************************************************************"
130 PRINT"**********************     W E L C O M E     ************************"
140 PRINT"*********************************************************************"
150 PRINT:PRINT:PRINT
155 FOR X=400 TO 1000 STEP 10:SOUND X,.5::NEXT X
160 PRINT "    You are welcome to use this IBM PC for any (well, almost any) purpose"
170 PRINT "you may desire. Please be courteous and allow time for other personnel"
180 PRINT "to access the machine (don't HOG IT!)."
190 PRINT:PRINT:PRINT
195 N$=""
200 INPUT "Please enter your full name ";N$
```

Lines 210 through 220 check the length of the name and shorten it to 30 characters if necessary. This shortening of the name is not necessary for correct execution of the program, but a standard length of 30 characters was decided upon so the log file would look organized when it is reviewed.

```
210 IF LEN(N$)>30 THEN N$=LEFT$(N$,30) :GOTO 230
220 N$=N$+STRING$(30-LEN(N$)," ")
```

Lines 230 and 240 retrieve the current date and time in string variables D$ and T$, respectively. Line 250 then creates a record in the string R$ that contains the 30 character name (N$), the date (D$), and the time (T$).

```
230 D$=DATE$
240 T$=TIME$
250 R$=N$+D$+" "+T$
```

Line 260 opens the file USERS.DAT so the record can be written identifying who signed on and when and the APPEND part of the OPEN command allows records to be added to the end of the file. This technique will eventually cause the file to become quite large, depending of course on the number of times the computer is used during any period of time. Line 270 writes the record (R$) and closes the USERS.DAT file.

```
260 OPEN "USERS.DAT" FOR APPEND AS #1
270 PRINT #1,R$ :CLOSE #1
```

Lines 280 through 300 thank you for logging on and advise you that the MENU will appear next. The SYSTEM command in line 490 terminates the BASIC program and returns control to DOS. Then, the AUTOEXEC.BAT file can contain commands to initiate the batch driven-menu system previously described in Chapter 4 in this book.

Lines 310 through 330 play a little tune to again "dress up" this program; you can remove the tune if desired.

```
280 FOR Z=1 TO 24:PRINT:NEXT Z
290 PRINT "Thank you . . . you will now be signed on at MENU level . . . good luck."
300 PRINT:PRINT:PRINT:PRINT
310 SCALE$="CDEFGAB":PLAY "O0 C;"
320 FOR X=1 TO 6:PLAY "L64; >XSCALE$;":NEXT X
330 FOR X=1 TO 6:PLAY "L64; >XSCALE$;":NEXT X
490 SYSTEM
```

Figure 7.6 shows an example of the WELCOME screen as it appears to the user as his or her name is requested. Note that once the user has logged onto the computer the message `Thank you . . . you will now be signed on at MENU level . . . good luck` from line 290 is displayed before the program returns to DOS (line 490).

Figure 7.7 is a copy of the sample USERS.DAT file. Note that because each name is forced to exactly 30 characters, the date and time values line up in a neat and orderly fashion for later review. This also enables you to write another BASIC program to analyze

Figure 7.6 **Initial Welcome Screen Display**

```
******************************************************************
*********************        W E L C O M E     *******************
******************************************************************

    You are welcome to use this IBM PC for any (well, almost any) purpose
you may desire. Please be courteous and allow time for other personnel
to access the machine (don't HOG IT!).

Please enter your full Name ?
```

Figure 7.7

Sample USERS.DAT File Contents

```
C>type users.dat
Tom Dolittle                     02-03-1988 15:05:43
Mary Jones                       02-03-1988 15:10:15
```

the USERS.DAT file for information about which users are the most frequent on any given PC and what time of day is the most popular with users.

Program Listing 7.17 gives the entire SIGNON.BAS BASIC program discussed in this section. You can use this program as a model to develop a more specific program to satisfy your operational requirements. Additional capabilities could be added to check for only certain names and to allow only specified users on any particular PC; however, be aware that *no* method is foolproof against a knowledgeable user. Thus, although you could incorporate password checking into the SIGNON.BAS file, a knowledgeable user could simply place a copy of DOS in drive A as he or she powered on the computer and could take control of the system, bypassing a previously created AUTOEXEC.BAS file that calls the SIGNON.BAS program.

Program Listing 7.17

SIGNON.BAS Program Listing

```
10 '************************************************************************
20 '**    SIGNON PROGRAM FOR USERS........as of 2-2-88
30 '**
40 '**  The purpose of this program is to maintain a log of all users who
50 '**  access and use this IBM PC. The user is prompted to enter a name,
60 '**  and a record is written to the file USERS.DAT containing the name
70 '**  (the first 30 characters only), the date, and the time of signon.
80 '**
90 '**  ...LAST CHANGE...2/3/88...Tom Dolittle
100 '*************************************************************************
```

```
119 CLS
120 PRINT "*********************************************************************"
130 PRINT "**********************     W E L C O M E     ************************"
140 PRINT "*********************************************************************"
150 PRINT:PRINT:PRINT
155 FOR X=400 TO 1000 STEP 10:SOUND X,.5:NEXT X
160 PRINT "    You are welcome to use this IBM PC for any (well, almost any) purpose"
170 PRINT "you may desire. Please be courteous and allow time for other personnel"
180 PRINT "to access the machine (don't HOG IT !)."
190 PRINT:PRINT:PRINT
195 N$=""
200 INPUT "Please enter your full name ";N$
210 IF LEN(N$)>30 THEN N$=LEFT$(N$,30):GOTO 230
220 N$=N$+STRING$(30-LEN(N$), " ")
230 D$=DATE$
240 T$=TIME$
250 R$=N$+D$+" "+T$
260 OPEN "USERS.DAT" FOR APPEND AS #1
270 PRINT #1,R$ :CLOSE #1
280 FOR Z=1 TO 24:PRINT:NEXT Z
290 PRINT "Thank you . . . you will now be signed on at MENU level . . . good luck."
300 PRINT:PRINT:PRINT:PRINT
310 SCALE$="CDEFGAB":PLAY "O0 C;"
320 FOR X=1 TO 6:PLAY "L64; >XSCALE;":NEXT X
330 FOR X=1 TO 6:PLAY "L64; <XSCALE;":NEXT X
490 SYSTEM
```

Logging Menu Selections

It is quite important for a good manager to know what software is the most popular with his or her employees and what software is not being used. This information could help to eliminate unnecessary purchases of expensive software packages that have no useful purpose to an organization. A simple method can be devised to log what software is being used when a batch menu system is used to both control and simplify access to the software.

A batch menu program previously described in Chapter 4 used the DOS change directory (CD) command to change to different subdirectories and execute various software products such as spreadsheets and word processors. The batch files in the menu program were named 1.BAT, 2.BAT, and 3.BAT; they corresponded to the menu selections (for example, choosing 1 gets the spreadsheet, choosing 2 gets the word processor, and so on).

A simple addition to two of these batch files is shown in Program Listing 7.18. Notice line 3 in each modified program where the ECHO command is located. Each line places the current date and time and the words *SPREADSHEET* or *WORD PROCESSOR,* respectively, in a file called LOG.DAT.

Program Listing 7.18

Modified 1.BAT and 2.BAT Programs

Modified 1.BAT Program

```
1:*echo off
2: cls
3: echo ? DATE$,TIME$,"SPREADSHEET" : SYSTEM ¦ BASIC ¦ FIND /V "?" ¦ FIND /V "Ok" >> LOG.DAT
4: cd \123
5: lotus
6: cls
7: cd \batutil
8: type mainmenu.txt
*
```

Modified 2.BAT Program

```
1:*echo off
2: cls
3: echo ? DATE$,TIME$,"WORD PROCESSOR" : SYSTEM ¦ BASIC ¦ FIND /V "?" ¦ FIND /V "Ok" >> LOG.DAT
4: cd \mm
5: mm
6: cls
7: cd \batutil
8: type mainmenu.txt
```

A close examination of the ECHO command in both programs shows that it is actually several commands on one line. The question mark (?) is the shortest way in BASIC to issue a PRINT command, and DATE$ and TIME$ are BASIC strings that contain the current date and time. The double quotes around "SPREADSHEET" indicate that it is a string to BASIC and the colon (:) and SYSTEM command are also BASIC symbols and commands. The vertical line (¦) symbol is a DOS piping and filter symbol that says "take everything passed from the left of this symbol and send it into the command on the right of this symbol." In this situation, the BASIC interpreter is called and the question mark, DATE$, TIME$, and string variable "SPREADSHEET" are passed to it just as if they were typed in from a program or interactive session. The colon (:) and SYSTEM command are also passed to BASIC, and BASIC interprets all this information as "PRINT the current date, time and the string SPREADSHEET, then exit BASIC to the operating SYSTEM."

The next vertical bar (¦) symbol accepts the output from BASIC (remember that we printed some things) and passes this output to the first DOS FIND command. This FIND COMMAND uses the /V parameter to exclude any line passed into it that contains the string enclosed in the double quote marks. In this case every string passed to the FIND command is searched for a question mark, and only strings that *do not* have question marks are passed out of the FIND command to the next filter (vertical bar) symbol.

The next FIND command filters out all lines that contain the Ok symbol, and the output is passed to a file called LOG.DAT. Note that the DOS piping symbol greater than (>) is shown twice in this example. Using the greater than symbol twice when sending output to a file is the same as APPENDING to the end of the file, thereby retaining what was previously in the file and keeping a log of each access of a SPREADSHEET or WORD PROCESSOR.

Based upon the preceding program lines, if you enter 1 from the previously described menu system, the modified 1.BAT file is run, causing the current date, time, and the string "SPREADSHEET" to be appended to the file LOG.DAT. Then the batch program changes the directory to 123 and initiates the LOTUS program. Similarly, the entry of 2 from the menu system would invoke the execution of the modified 2.BAT file, causing the current date, time, and the string "WORD PROCESSOR" to be appended to the LOG.DAT file. Then that batch program changes the directory to MM and invokes the execution of a program file named MM.

Figure 7.8 contains a sample output from the LOG.DAT file. While this file only contains the date and time each program was accessed, it is a simple matter to modify the programs previously illustrated in Program Listing 7.18 to log both the time of access and the time of exit from each program. Then, if this information is contained on the LOG.DAT file, you could create a BASIC program to use the contents of that file to determine the duration of each session.

Determining Session Duration

In discussing the changes required to log the starting and ending time of each session, we will use the contents of the modified 1.BAT and

Figure 7.8

Sample Output from LOG.DAT File

```
C>type log.dat
02-03-1988   15:08:55   WORD PROCESSOR
02-03-1988   15:09:10   SPREADSHEET
02-03-1988   15:09:25   SPREADSHEET
02-03-1988   15:09:38   WORD PROCESSOR
02-03-1988   15:10:24   SPREADSHEET

C>
```

2.BAT files previously given in Program Listing 7.18 as a point of reference. Because the computation of a session's duration requires knowledge of both the start and end times, the string in line 3 of each program should be changed. This change should indicate that that date and time represents the start of a spreadsheet or word processing session. Thus, the string `"SPREADSHEET"` could be changed to `"S SPREADSHEET"` with the first S now placed in a field where it would indicate the start of the spreadsheet session. Similarly, the string `"WORD PROCESSOR"` in line 3 of the modified 2.BAT program could be changed to `"S WORD PROCESSOR"`.

To denote the end of each session, you could insert a new line into each program prior to line 8. This line would be very similar to line 3, with the only change in the string contained on that line. Here you would change the string `"SPREADSHEET"` in the 1.BAT program to the string `"E SPREADSHEET"`, where an E would indicate that the date and time of the record written onto the LOG.DAT file reflect when the session ended. Similarly, a copy of line 3 of the 2.BAT program—when moved to line 8—would have the string `"WORD PROCESSOR"` replaced by the string `"E WORD PROCESSOR"`. Once these modifications are made, the LOG.DAT file will contain the starting and ending times of each spreadsheet and word processing session, providing you with the information necessary to determine the duration of each session.

ELIMINATING DISK FRAGMENTATION

Both hard and floppy diskettes lose read and write efficiency as the media is used. This loss of efficiency is due to the manner in which DOS adds and deletes information.

After a disk or diskette is formatted, data initially written onto the media is stored in sector sequence by track. That is, if there are

8 sectors per track and a file to be saved will fill 12 sectors, it will fill the first track and the first 4 sectors of the second track. As additional files are saved to disk, DOS searches for sequentially available sectors and stores data in the sequence of sectors that are available for use. Thus, if a second file to be saved contains 20 sectors of information, it would be stored in the last 4 sectors of the second track, all 8 sectors of the third track, and all 8 sectors of the fourth track. After the second file is saved, we will assume that you then wish to delete the first file.

When you issue a DOS command to delete or erase a file, DOS does not actually delete the file from the media. Instead, DOS sets a bit in an entry in a File Allocation Table on disk. Once this bit is set, it indicates that the file and all of the sectors previously used to store the file are available for DOS to store new information.

Now suppose you wish to store a file consisting of information that requires 10 sectors for storage. DOS would use the 8 sectors on the first track, which just became available, and 2 sectors on the fifth track, resulting in the fragmentation of the file.

As a storage media is used, the files on the media normally become more and more fragmented. This is due to normal working activities that include the deletion and storage of data and cannot be avoided in regular day-to-day operations. Unfortunately, as files become fragmented, the time to retrieve or store data on the media increases. This is because the read/write heads of the storage device have to move back and forth to locate fragments of data on different tracks. In addition to increasing data storage and retrieval times, the extra wear and tear on the electromechanical disk heads most likely shortens the life of the storage unit.

DISKCOPY Versus COPY

The simplest method of eliminating fragmentation on a diskette is to create a new copy of the disk. Although you might be initially tempted to use the DISKCOPY command, because it automatically formats a disk prior to copying data onto the media, that command should not be used. The reason DISKCOPY is invalid for eliminating diskette fragmentation is that it makes an exact copy of a disk, copying data sector-by-sector. Thus, if your diskette is fragmented, the new diskette created by the DISKCOPY command will also be fragmented.

If you use the COPY command, each file copied from a source to target diskette is placed on the target diskette in sector sequence, regardless of its fragmentation on the source diskette. Thus, the COPY command is the key to removing diskette fragmentation.

Program Listing 7.19 contains the listing of a batch program that can be used to eliminate diskette fragmentation. Line 1 disables the echoing of lines in the program when the program executes while line 2 clears the screen. Lines 3 through 5 simply display the message `UNFRAGMENT DISK` on the second line of the display screen.

Program Listing 7.19

UNFRAGMT.BAT Program

```
 1:*echo off
 2: cls
 3: echo .
 4: echo .    UNFRAGMENT DISK
 5: echo .
 6: IF "%1" == "F" GOTO :FMT
 7: GOTO :CPY
 8: :FMT
 9: FORMAT B: %2 %3 %4 %5
10: :CPY
11: COPY *.* B:
12: echo .
13: echo .   All Done.....
```

Line 6 includes the IF command to test if the replaceable parameter F was entered in the command line when the command UNFRAGMT was issued. If the letter F was included in the command line, the program branches to the label :FMT in line 8, and the FORMAT command in line 9 is then executed.

The FORMAT command in line 9 uses the B drive as the destination drive; however, you could use another drive in your routine or a replaceable parameter to specify the drive. The replaceable parameters %2, %3, %4, and %5 in line 9 enable you to include format parameters in the UNFRAGMT command line after you specify the letter F, if you so desire. If the letter F was not specified, the program will branch to the label :CPY in line 10, after which the DOS COPY command with the asterisk (*) used as global characters copies all files from the default drive to drive B. Finally, the ECHO commands in lines 12 and 13 of the program simply verify the completion of the batch program.

Figure 7.9 illustrates the execution of the batch file when the target diskette is to be formatted prior to the files being copied to that disk. As shown in the figure, the formatting process displays the `Insert new diskette for drive B:` and the `and strike ENTER when ready` messages. Once the formatting process is completed, you must also answer the `Format another (Y/N)?` prompt before the COPY process begins. To automate this step you could create a file with the character *N* in it and issue the FORMAT command in line 9 with input from the file containing the character *N*. Although this action will automate the format operation, it will also restrict you: you could either predefine the parameters to incorporate into the FORMAT command by including those parameters in the file with the character *N*, or you could accept the default parameters associated with the FORMAT command. See Figure 7.10.

Several other restrictions are worth noting with this particular implementation of the UNFRAGMT.BAT file. The FORMAT.EXE

Figure 7.9 **Execution of UNFRAGMT.BAT Using the F Option**

```
.
.   UNFRAGMENT DISK
.
Insert new diskette for drive B:
and strike ENTER when ready

Format complete

    730112 bytes total disk space
    730112 bytes available on disk

Format another (Y/N)?n

PAY87.DAT
FORMAT.BAT
PRTLABEL.BAK
PRTLABEL.BAT
UNFRAGMT.BAT
FORMAT.COM
UNFRAGMT.BAK
        7 File(s) copied
.
.   All Done...
A>
```

Figure 7.10 **Execution of UNFRAGMT.BAT Without Specifying the F Option**

```
.
.   UNFRAGMENT DISK
.
PAY87.DAT
FORMAT.BAT
PRTLABEL.BAK
PRTLABEL.BAT
UNFRAGMT.BAT
UNFRAGMT.BAK
        6 File(s) copied
.
.   All Done...
A>
```

command file must be available in the current directory or through the current path setting. The B drive is used in this example as the destination drive, although any drive could be written into your routine. The UNFRAGMT.BAT file must be in the current directory or available through the current path setting. As a last note, the COPY command does not support copying subdirectories and files within those subdirectories.

Modifications to Consider

Although the larger hard disk storage capacity makes that device a better candidate for eliminating fragmentation, we have purposely avoided mentioning this storage medium until now. This omission is due to the serious consequences that can result from formatting a hard disk if a prior backup operation is not done correctly. For this reason, you should manually perform a backup of your hard disk prior to reformatting the disk. Once the backup operation using the DOS BACKUP command is completed correctly, you can then reformat your hard disk. You must have a version of DOS on a diskette to install from a floppy-diskette drive, so that you can use the DOS RECOVER command to place the files on your hard disk in sector order sequence.

From personal experience, reformatting a 20M-byte hard disk after a year of active use appeared to double the data retrieval and file writing speed of the disk. Of course, your actual experience will depend upon the degree of fragmentation of the data on your disk.

EFFECTIVE USE OF RAM DISKS

There are several situations in which random access memory (RAM) can be effectively used as an additional diskette drive. First, even if your system has a hard disk, you may be running out of room on that disk and may wish to use a diskette to store data files you do not frequently use. Secondly, if your system is limited to two floppy-diskette drives, the ability to use RAM as an additional diskette drive may enable you to eliminate or reduce the "floppy shuffle." In this section we will examine the use of a batch file that works in conjunction with the DOS CONFIG.SYS file to use RAM as an additional diskette drive.

Assume you have a dual diskette system and upon occasion use a word processor in conjunction with a dictionary of legal terms. Further assume that the size of the word processor and legal dictionary require each to be resident on separate diskettes. If your word processor diskette previously had the DOS system files transferred to it, it is self-booting and you could install it in drive A when you power on your computer. Unfortunately, if you wish to have a separate diskette for your letters and memorandums, you must swap the legal dictionary and data diskette into and out of drive B whenever you wish to retrieve or save data files or check the spelling of your memorandums. In this type of situation the use of RAM as drive C could alleviate a large portion of the floppy shuffle problem.

If you have DOS 3.0 or a later version of the operating system, you can easily use the VDISK (virtual disk) program that is contained on your system diskette. Assuming you previously transferred DOS system files to your word processing program, you can set up a RAM disk each time you boot your word processor, by creating a configuration file that will set up a device driver for the DOS VDISK program. To do this, at the DOS prompt enter the command:

```
COPY CON: CONFIG.SYS
```

Next, to create a 320K RAM disk that functions as drive C you would enter:

```
DEVICE=VDISK.SYS 320
```

Once you press the F6 key or the CTRL-Z key combination, the CONFIG.SYS file will be saved on your word processing diskette. After this, copy the VDISK.SYS file from your DOS diskette to the word processing diskette. Then, each time you boot your word processing program disk, the CONFIG.SYS file will initiate a 320K RAM disk that functions as drive C.

Controlling Drive C

Now that a RAM disk will be set up as drive C, we need a method to automate the transferring of the legal dictionary to that drive.

Because many legal dictionaries permit you to add new terms to the spelling checker, our batch file should also permit the updated dictionary to be saved back onto the original diskette. In addition, if we are going to create a batch file to automate the previously mentioned tasks, we can also initiate the word processing program and control the insertion and removal of our data diskettes by the use of the batch file.

Program Listing 7.20 lists the batch file that accomplishes all of the previously mentioned functions. Because we have assumed that it would be advantageous to automate the entire process, we have created an AUTOEXEC.BAT file on the word processing diskette that will be invoked if we power on the computer with that diskette in drive A. If the computer was previously powered on, you can simply enter the command AUTOEXEC to execute this file.

Program Listing 7.20

The AUTOEXEC.BAT File

```
COPY CON: AUTOEXEC.BAT
ECHO OFF
ECHO PLACE LEGAL DICTIONARY DISKETTE IN DRIVE B
PAUSE
IF EXIST B: LEGAL.XXX GOTO :OK
ECHO WRONG DISK INSTALLED--PROGRAM TERMINATING
GOTO :END
:OK
COPY B: *.* C:
ECHO PLACE DATA DISKETTE IN DRIVE B
PAUSE
WP
CLS
ECHO PLACE LEGAL DICTIONARY DISKETTE IN DRIVE B
PAUSE
COPY C: *.* B:
```

```
ECHO LEGAL DICTIONARY UPDATE COMPLETE
:END
```

Line 1 turns off the display of commands as they are executed in the batch file, and line 2 displays the message PLACE LEGAL DICTIONARY DISKETTE IN DRIVE B. The PAUSE command stops the execution of the program until a key on the keyboard is pressed. In line 4, the IF EXIST command checks for the presence of the legal dictionary file, which we named LEGAL.XXX for this example. If this file is not found on the diskette in drive B, line 5 displays an error message and line 6 causes a branch to the end of the program.

If the correct file is on the diskette in drive B, the program branches to the label OK. Next, the COPY command copies the contents of the legal dictionary to the RAM disk, which is drive C. In line 9, the ECHO command prompts you to place a data diskette in drive B, and line 11 initiates the word processing program (named WP for this example).

Once the word processing program is terminated, control returns to the batch file listed in Program Listing 7.20. After the CLS command clears the screen, the ECHO command prompts you to place the legal dictionary diskette back into drive B, because you may have added terms to the dictionary and should therefore update the real diskette. This update is handled by the last COPY command in the file, which copies the contents of the RAM disk onto the diskette in drive B.

EXIT CODE PROCESSING

There are several MS-DOS commands that set an exit code, either after they finish executing or to terminate their execution due to various problems that can abort the command. These exit codes are also referred to as the command's error level and can be used within a batch file to determine both the cause of a problem as well as a mechanism to display an appropriate message to the computer operator.

Even if you are executing an unattended batch file, the concept presented in this section for displaying messages associated with an error level or exit code can be used to indicate the reason that a sequence of commands aborted. Then, when a computer operator returns and examines the display, detailed information is available to assist in correcting whatever problem previously arose.

Error-Level Processing

Table 7.5 lists the error level or exit codes associated with five MS-DOS commands. Note that the TAPE command is not included in IBM's version of DOS and is only applicable for compatible computers that have a tape backup unit installed in the system unit or externally cabled to a controller card installed in their system unit.

As an example of the use of error-level processing in a batch file, assume you wish to automate the operation of the BACKUP command. Since the format of this command can become quite complicated for the novice DOS user, you might consider creating a command called ARCHIVE, which executes a predefined version of the BACKUP command. In doing so you could perform, within the ARCHIVE.BAT file, error-level processing that could provide the novice user with information concerning the execution of the BACKUP command. Program Listing 7.21 shows the creation of the batch file named ARCHIVE that, in effect, executes a predefined BACKUP command.

Program Listing 7.21 **Creating the ARCHIVE.BAT File**

```
COPY CON: ARCHIVE.BAT
ECHO OFF
```

Table 7.5 **Exit/Error-Level Codes**

Command	Code	Meaning
BACKUP	0	Normal completion
	1	No files found
	2	File sharing conflict
	3	Terminated by CTRL-BREAK
	4	Terminated because of an error
REPLACE	0	Normal completion
	1	CTRL-C abort or user answered A to an `Abort, Retry, Ignore` prompt
	2	File not found
	3	Path not found
	4	Path too long
	5	Access denied
	8	Insufficient memory
	11	Command line error
	15	Invalid drive
	22	Invalid MS-DOS version
RESTORE	0	Normal completion
	1	No files found
	2	File sharing conflict
	3	Terminated by CTRL-BREAK
	4	Terminated because of an error
TAPE	0	Normal completion
	1	No files found
XCOPY	0	Normal completion
	1	No files to copy
	3	Terminated by CTRL-C, CTRL-BREAK, or answered A to an `Abort, Retry, Ignore` prompt
	4	General error termination

```
REM THIS BATCH FILE BACKS UP ONLY MATCHING
REM FILES FROM DRIVE C UNDER THE DBASE
REM DIRECTORY WHOSE FILE EXTENSION IS
REM DAT THAT WERE MODIFIED SINCE THE
REM PREVIOUS BACKUP
PAUSE INSERT DISK LABELED XXX INTO DRIVE A
BACKUP C:\DBASE\ *.DAT A: /M
IF ERRORLEVEL 0 ECHO NORMAL COMPLETION
IF ERRORLEVEL 1 ECHO NO FILES FOUND TO BACKUP
IF ERRORLEVEL 3 ECHO USER TERMINATED PROGRAM WITH CTRL-C
IF ERRORLEVEL 4 ECHO ERROR CONDITION EXISTS
```

As indicated by the REM statements in the listing, this batch file checks all files on drive C with .DAT extensions that are located in the DBASE directory. If these files on drive C are also on the disk in drive A and were modified since the last backup operation occurred, they will be copied to drive A, in effect, updating a previously performed backup operation. As you will note from the use of the BACKUP command, its parameter settings, while not as complex as possible by other uses of the command, may still be too complex for a novice to specify correctly. Thus, its use in a batch file can be expected to help users back up data easily.

The last five lines of the ARCHIVE.BAT program use the DOS IF command to process all possible error-level or exit codes that can occur through the use of the BACKUP command. Thus, if the BACKUP command was performed successfully, the message `NORMAL COMPLETION` would be displayed on the user's screen, whereas an appropriate error message would be displayed if the BACKUP operation was not successfully completed.

As you examine the construction of the ARCHIVE.BAT program, note the message placed in the PAUSE command that is displayed prior to execution of the BACKUP command. In this example, the string "XXX" would be replaced by the label of the disk used for the backup operation. As an additional mechanism for ensuring the successful operation of the ARCHIVE command, you could use the DOS IF command to check for the existence of a known file on the diskette. Then, if the file was not found you could branch to a label entry point in the program that would tell the operator that he or she inserted the wrong diskette into drive A and should enter the command ARCHIVE to start over. Thus, after the PAUSE command you could insert the line:

```
IF NOT EXIST XXX.DAT GOTO :DSKERROR
```

where the string "XXX" would represent the name of a file that should be on the backup diskette. You could then place the label :DSKERROR at the end of the file previously listed in Program Listing 7.21, following that line by the following lines:

```
ECHO WRONG DISKETTE INSERTED IN DRIVE A
ECHO INSERT DISKETTE LABELED "XXX"
ECHO THEN REENTER COMMAND ARCHIVE
```

As indicated in this section, a combination of error-level testing and file testing can be used to provide the user with a better indication of the reason why intended operations fail. You can combine the utilities presented in this section with other batch files, in order to perform more complex functions.

DATE AND TIME ACCESS

Although DOS is very comprehensive, it does not provide an easy mechanism to access the date or time from within a batch file. In this section we will examine how we can extract the current date from the DOS DATE command. Once this is accomplished, you can use the day of the week or the actual numeric date in your batch file. Although we will focus our investigation here on the extraction of the current date, the concept also applies to the current time through the use of the TIME command.

The DATE Command

The key to accessing the current date is the DOS DATE command. If you type the command DATE, your system's response will be similar to the following, but will show the current date of the day on which you issue the command.

```
Current date is Fri 3-04-1988
Enter new date (mm-dd-yy):
```

If you have an AUTOEXEC.BAT file, you can add the following line to that file:

```
ECHO  MORE  DATE>>MYDATE.BAT
```

When the preceding line executes, the MORE command provides the carriage return required by the DATE command to terminate the execution of that command. That is, if you enter a carriage return, the DATE prompt `Enter new date (mm-dd-yy):` does not receive a new date, so the new date reverts to the current date value. The two-line output of the DATE command is therefore appended to the file MYDATE.BAT.

Now we can add the command MYDATE at the end of the AUTOEXEC.BAT file. When this line is executed by DOS, the operating system will attempt to execute a .COM, .EXE, or .BAT file whose filename is CURRENT. This is because DOS assigns the first line's value, `Current`, in the MYDATE file, `Current date is Fri 3-04-1988`, to the replaceable parameter %0, `date` to the replaceable parameter %1, `is` to the replaceable parameter %2, `Fri` to the replaceable parameter %3, and the actual numeric date, `3-04-1988` to the replaceable parameter %4.

Because DOS will attempt to execute a file named CURRENT, let us create a batch file with that name. This file will extract the current day of the week and then execute another program called NEXTPROG only once a day. You can substitute the name of the actual program you desire to execute. Further assume that you only wish to execute NEXTPROG on Fridays. Program Listing 7.22 illustrates the creation of the CURRENT.BAT file that accomplishes the previously mentioned functions.

Program Listing 7.22

Creating CURRENT.BAT

```
COPY CON: CURRENT.BAT
ECHO OFF
IF !%DAY%==!%3 GOTO :END
SET DAY=%3
IF NOT %3==Fri GOTO :END
NEXTPROG
:END
```

The DOS environment gives the CURRENT.BAT program the ability to extract the required data. The environment is a series of ASCII strings that provides a mechanism for communicating between programs and can be examined and/or modified by other programs. Examples of DOS commands that place strings of characters into the environment include PATH, PROMPT, and SET.

After the first line in the file disables the echoing of commands as they are executed, the second line tests to see if the environment

%DAY% variable has the same value as the replaceable parameter %3. If so, the program was previously executed during the current day and control is passed to the label END. If the environment variable named DAY does not equal the current day of the week, control is passed to line 4 in the program. Here the SET command sets the variable DAY equal to the current day of the week, and the next line tests the day of the week against Fri. If the day of the week is not Friday, a branch to the label END occurs. However, if the day of the week is Friday, the fifth line in the file executes and invokes the file NEXTPROG.

Time Processing

The DOS TIME command accesses the current time. If you type the command TIME, your system's response will be similar to the following, with the actual current time dependent upon when the command is executed.

```
Current time is 13:18:22.28
Enter new time:
```

By incorporating the following line into your file, you can append the two-line response of the TIME command to the file MYDATE.

```
ECHO  MORE  TIME>>MYDATE.BAT
```

Then, if you added the command MYDATE to your file, DOS will attempt to execute a .COM, .EXE, or .BAT file whose filename is CURRENT. When this occurs, DOS will assign `Current` to the replaceable parameter %0, `time` to the replaceable parameter %1, `is` to the replaceable parameter %2, and the time resulting from the previous execution of the TIME command to the replaceable parameter %3. Because DOS attempts to execute a file named CURRENT, you could modify the previously created CURRENT.BAT file illustrated in Program Listing 7.22 to perform time processing. Unfortunately, because both the DATE and TIME commands respond with a prompt message starting with the word *Current,* normally you can only construct one batch file to process and extract either the day or time.

However, because each subdirectory can have a file with the same name as a file in a different subdirectory, you could place one CURRENT.BAT file in one subdirectory and a second file in a different subdirectory. Doing so would enable you to create a batch program that switches directories when you create the MYDATE.BAT file, resulting in one MYDATE.BAT file in one subdirectory containing the output of the DATE command, while a second MYDATE.BAT file located in a different subdirectory would contain the output of the TIME command. Then, after DOS automatically executes the CURRENT.BAT program located in the default subdirectory, your main batch file could change subdirectories and issue the command CURRENT to execute the second CURRENT.BAT file.

As indicated in this section, you can use several relatively simple tricks for DOS batch files to extract the day of the week and/or the numeric value of the date from the DOS DATE command or the time from the DOS TIME command. Then, you can use the extracted values within a batch file to govern how data processes or time or date stamp information.

DETERMINING THE SIZE OF A DIRECTORY

There are many occasions when you move files from a hard disk to a floppy diskette. Sometimes the files to be moved are simply data files that you will use with another computer. Other times you want to move an entire subdirectory of data files and associated software to another location. Anyone who has ever duplicated files onto a floppy disk has most likely used the `COPY *.* A:` command, and has probably more than once seen the error `Disk full` appear on the screen. This error message indicates that the floppy disk cannot hold any additional files. At this point you must copy all remaining files in the directory to one or more additional floppy diskettes.

It would save time and effort to know, before you started a COPY operation, whether all of the files in the directory being copied will fit onto the floppy disk you are writing to. In this section we will develop a batch file to invoke a BASIC program to tell us whether or not all the files in a particular subdirectory will fit on a particular floppy disk. If all files will fit then a simple `COPY *.* A:` command will suffice for the file movement. However, what about the situation when files will not fit? Knowing this information ahead of time will help us to plan the movement of files in groups, such as `*.DAT` or `PROG????.*`. In addition, when you know how much storage the files in a subdirectory require, you can determine if you have a sufficient number of formatted diskettes to accomplish the COPY operation.

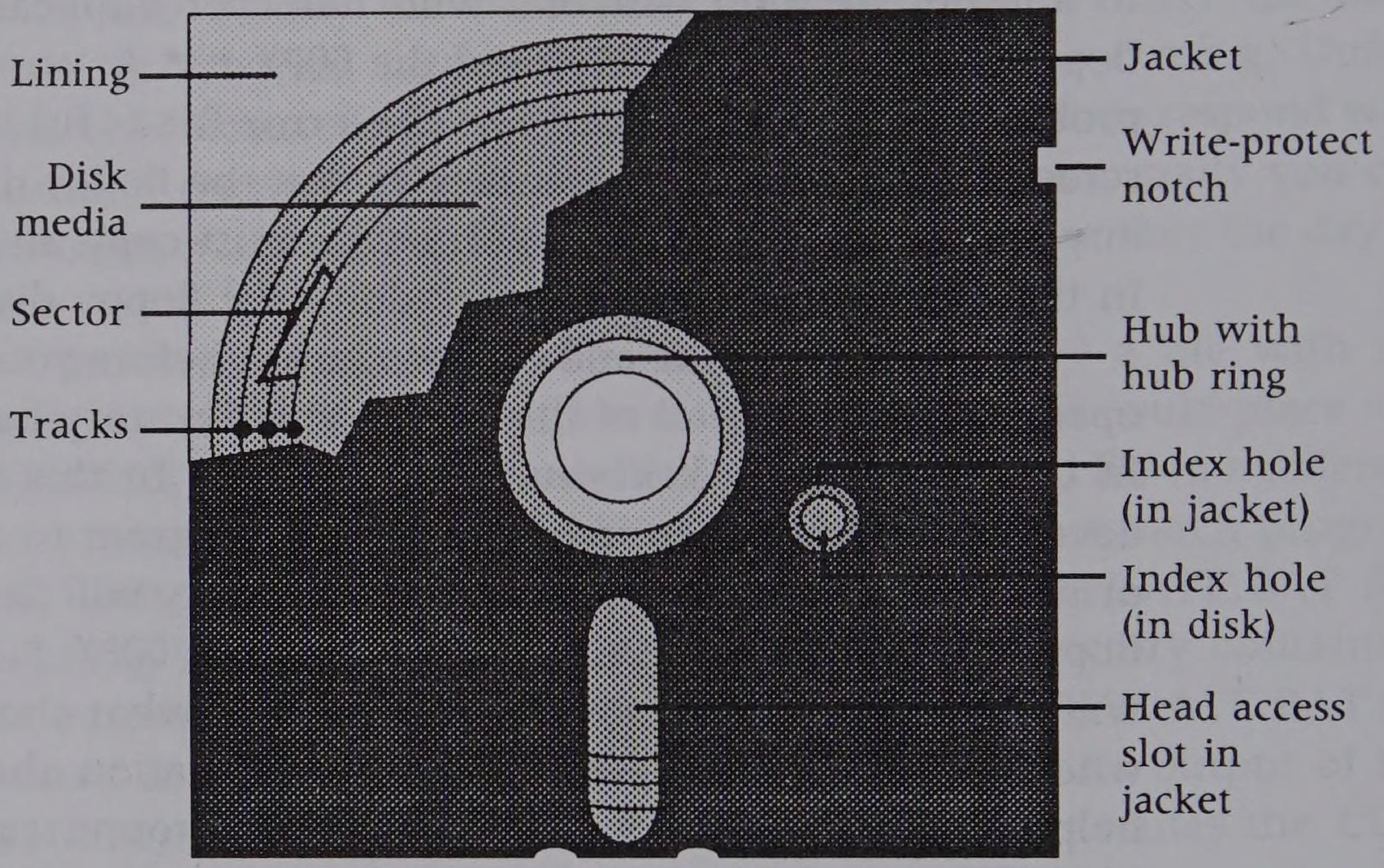
Diskette Capacity

To begin, we will explore some of the types of diskettes and the way data is written on them. The original IBM PC introduced in 1981 used DOS 1.0 and supported only single-sided floppy diskette drives. Shortly afterwards, IBM revised the operating system to version 1.1, which added support for diskette drives that could write on both sides (double-sided diskette drives).

Figure 7.11 illustrates the cross section of the original 5¼ inch diskette used in IBM PCs and compatible computers. The diskette is a circular sheet of Mylar plastic, coated with ferric oxide, that is

Figure 7.11

Cross-Section of a 5¼ Inch Floppy Diskette


Lining
Disk media
Sector
Tracks
Jacket
Write-protect notch
Hub with hub ring
Index hole (in jacket)
Index hole (in disk)
Head access slot in jacket

enclosed in a square protective jacket. The diskette rotates clockwise within its jacket.

Before you can use a blank diskette, you must format it. That is, you must make the computer write data onto all tracks on the diskette in such a way that it can later identify precisely which part of the diskette the heads that perform read/write operations are under. During this formatting process the computer divides the recording space on the diskette into concentric circles called tracks. These tracks begin at the outer edge of the diskette with the number 0 and move to the inner edge where track number 39 is written. This gives a total of 40 tracks on a single-sided diskette. If the diskette is double-sided, there are 40 tracks on each side of the diskette. The upper and lower tracks are numbered identically, but are differentiated by the number of the head that reads or writes them (0 for the upper, 1 for the lower head).

The computer identifies the start of each track as the point at which the index hole in the diskette becomes aligned with the index hole in the jacket and allows a beam of light to pass through both holes. A photocell detects the light beam and generates an index signal to the computer. At this time the computer writes a special pattern called the index mark onto the disk.

The computer next divides each track up into sectors by writing a synchronization pattern consisting of the track number, head number and sector number, followed by the appropriate number of bytes of a special character that denotes an empty data space (usually this special character is an E5 hex). After the last data byte, the computer writes two check bytes that enable the detection of reading errors. The computer repeats this sequence of sector ID, data bytes, and check bytes as many times as there will be sectors on the track.

Note that the maximum number of bytes per track is determined partly by the diskette rotation speed and partly by the encoding method employed by the disk controller. Within these limits, however, the number of bytes per sector and the number of sectors per track can vary according to the requirements of the operating system. Diskettes that are formatted by the process previously described are said to be *soft sectored.*

The original IBM PC support for single-sided diskette drives used 40 tracks on a disk with 8 sectors per track. This resulted in a storage capacity of 40 tracks times 8 sectors/track times 512 bytes/sector, or 163,840 bytes of information. When double-sided drives were introduced, their storage capacity was 2 sides times 40 tracks/side times 8 sectors/track times 512 bytes/sector, which effectively doubled the storage capacity to 327,680 bytes of information.

When IBM announced the IBM PC XT, it also introduced a new version of DOS, Version 2.0. Under this version and those that followed, a 5¼ inch diskette could be formatted with 9 sectors per track. Thus, a double-sided diskette could have a storage capacity of 2 sides times 40 tracks/side times 9 sectors/track times 512 bytes/sector, or 368,640 bytes of information.

When IBM announced the PC AT this computer supported a 5¼ inch, half-height, double-sided diskette drive capable of storing 1.2M bytes on a diskette. This diskette drive is actually a high-capacity device capable of storing data on 80 tracks using 15 sectors per track. The separation between tracks is smaller than on a conventional 360K byte diskette, so special high-capacity diskettes must be used in this diskette drive. The storage capacity here becomes 2 sides times 80 tracks/side times 15 sectors/track times 512 bytes/sector, yielding 1,228,800 bytes of information (or 1.2M bytes).

Not one for ignoring change, IBM announced another machine—the portable computer—that changed the storage media from 5¼ inch to 3½ inch diskettes while introducing a new version of the operating system (DOS 3.2) to support the new media. These diskettes were formatted with 80 tracks per side, using 9 sectors per track. Thus, 2 sides times 80 tracks/side times 9 sectors/track times 512 bytes/sector yielded 737,280 bytes of information (or 720K bytes).

With the introduction of the PS/2 series of computers, IBM introduced high-capacity 3½ inch diskette drives that can hold 1.4M-bytes. The storage capacity here is obtained by writing 18 sectors per track on special high-capacity 3½ inch diskettes. The storage capacity now becomes 2 sides times 80 tracks/side times 18 sectors/track times 512 bytes/sector, yielding 1,474,560 bytes of information.

Available Storage Capacity

Armed with all this information about diskettes and space available on different formats, you might notice a discrepancy between the amount of space calculated on the diskette and the amount returned from a FORMAT command. As an example, in formatting a diskette on an IBM PC XT with a double-sided 360K byte drive, the format operation will normally display a message that 362,496 bytes were available. But our previous computations showed that we should have 368,640 bytes available if a default of 9 sectors per track is used under DOS 2.0 and later versions of the operating system. The 6,144 bytes have been used on the diskette by the *File Allocation Table (FAT)*. This is a table that keeps up with what files are on the diskette (filename) and where they are located (track and sector address). This is somewhat analogous to the table of contents in a book and helps the computer to locate any file quickly and accurately on the diskette by looking up its name in the FAT, finding the location on the disk and causing the read/write heads to move to the proper location to read the information.

Batch File Construction

Now, take a look at a batch file and BASIC program that can help you determine if a directory of information will fit onto a single diskette. Program Listing 7.23 shows a batch file called SIZE.BAT. Line 1 simply turns off the echo of commands to the screen. Line 2 issues the DOS directory command for any directory entered by the user. Here, the name of the directory is assigned to the %1 replaceable parameter.

Program Listing 7.23

SIZE.BAT Program

```
1:*echo off
2: dir %1 ¦ FIND ":" ¦ FIND /V "<DIR>" ¦ FIND /V "Directory" > dir.dat
3: basic sizeprog.bas
```

If you do not specify a directory with the SIZE command, the current directory is used. DOS piping is used to pass the output of the directory command to the first DOS FIND command that selects only lines from the directory output that contain a colon symbol (:). This command is used with a colon because any PC with a clock will automatically append a day and time. It adds a colon between the hour and minute and inserts the time into the file's directory entry. DOS piping is again used to pass the lines selected to a second FIND command, which eliminates all lines which are subdirectory listing (directories subordinate to the directory we are working with). Use the /V switch to locate all lines that do not contain the string "<DIR>". This output is then passed to a third FIND command, which eliminates a standard part of the directory command output, a line stating what directory is being displayed. Finally, the output from the directory command is passed to a file called DIR.DAT.

Line 3 of the SIZE.BAT file calls and executes a BASIC program called SIZEPROG.BAS. Program Listing 7.24 contains a listing of the SIZEPROG.BAS program that will actually compute the number of bytes contained in all files in the subdirectory we are working with.

Program Listing 7.24

BASIC Program to Determine File Space Requirements

```
10 OPEN "DIR.DAT" FOR INPUT AS #1
20 L$=""
30 D1#=368640!
40 SZ1=1024
50 LET SUM#=0
60 LINE INPUT #1,L$
70 IF EOF(1) THEN GOTO 130
80 LET N#=VAL(MID$(L$,13,9))
90 LET X#=INT((N#/SZ1)+.9999)
100 LET SUM#=SUM#+X#
110 LET TOT#=SUM#*SZ1
120 GOTO 60
130 CLOSE #1
140 IF TOT#<D1# THEN PRINT "Files will fit on DISKETTE" ELSE PRINT "not OK.."
150 IF TOT#<D1# THEN PRINT "Total space required = ";TOT#
160 IF TOT#<D1# THEN PRINT "Total space on disk  = ";D1#
```

```
170 SYSTEM
180 END
```

Line 10 opens the DIR.DAT file for input so that it can be read by the program. Line 20 sets a string variable (L$) to a null or empty value. Line 30 sets a double precision numeric variable called D1# to the value 368640, which is the amount of data that can be written on a double-sided 5¼ inch diskette with 9 sectors per track. Line 40 sets a numeric variable called SZ1 to the value 1024. Line 50 sets a variable called SUM to the value 0. Line 60 reads the first line from the DIR.DAT file and places it into the string variable L$. Next, line 70 tests if the end of file condition has been reached on the DIR.DAT file being read and when all records have been read, control is passed to line 130 of the program. Line 80 sets the numeric variable called N# to the value of the size of the file found in this record from the directory listing that was written to DIR.DAT. Here the MID$ function extracts 9 characters from the L$ string, beginning at the character position 13 in the string. Thus, line 80 extracts the byte count from each file record.

Line 90 divides the size of the file by SZ1 (1024), adds .9999 to the result, then converts the number to an integer, and truncates any excess over and above the whole number value generated. This whole number is the number of sectors required to write the file to a floppy disk. Line 100 increments the SUM# variable by the number of sectors, and line 110 computes the total number of bytes accumulated into the variable TOT#. Line 120 returns to line 60 to read the next record from the DIR.DAT file. Line 130 is executed when the DIR.DAT file is completely processed, resulting in the file being closed. Lines 140 through 160 check to see if the total bytes required for writing all files to the disk is less than the amount the disk will hold. An informative message displays to announce whether the files will fit on the diskette.

Figure 7.12 shows an example of the execution of the SIZE.BAT program. In this example, the SIZE command was entered without a subdirectory specification, and the current directory was generated and checked for total byte requirements.

Disk and Operating System Considerations

But wait a minute, why was the value 1024 used in the BASIC program as the sector size instead of 512 as previously discussed? In researching and planning the SIZE utility, I ran into some interesting situations with different diskettes, different operating system versions and different types of PCs. Figure 7.13 shows an example of how many bytes are required to write an 8 byte file to a 360K byte floppy diskette formatted on an IBM PC XT using DOS 2.1. Notice that a directory command was issued prior to creating the XXX.ZZZ file of 8 bytes. The original directory shows that 324608 bytes are free for storage, but after creating the 8 byte file, only 323584 bytes are free. The difference is 1024 bytes, implying that it took 1024 bytes on the disk to store an 8 byte file. This is contrary to the 512 byte discussion we had earlier in this chapter, but notice that 512 + 512 = 1024.

Figure 7.12

Executing the SIZE.BAT Program

```
SIZE
Files will fit on DISKETTE
Total space required =  56320
Total space on disk  =   368640

C>
```

Figure 7.13

Determining the Cluster Size

```
TVLTICKE DOC      2048   2-16-88   3:13p
SHARE7OC DOC      2560   3-10-88  11:55a
WAYNEBRO DOC      2560   3-11-88   9:42a
TELEBITM DOC      3072   3-11-88  11:23a
TAPEMEMO DOC      1536   3-11-88  12:32p
DOWDENME DOC      2560   3-12-88   3:08p
SHIFTCHA DOC      4096   3-12-88  12:09p
JACKPOLA DOC      2048   3-16-88   2:38p
GILHELD  DOC      2560   3-16-88   3:10p
       12 File(s)    324608 bytes free

C>copy con a:xxx.zzz
abcdef
^z
        1 File(s) copied

C>dir a:*.zzz

Volume in drive A has no label
Directory of A:\

XXX      ZZZ         8   3-16-88   3:29p
        1 File(s)    323584 bytes free

C>
```

Determining the Size of a Directory

The reason for using a sector size of 1024 bytes is the fact that when data is stored on the disk in a PC XT it is recorded in physical units known as *clusters.* Here a cluster is actually two sectors of information that the disk's read/write heads record and retrieve as a unit of 1024 bytes of data. Essentially, clusters are used on some diskettes and all hard disks, because they permit the FAT to address more information than if it directly addressed individual sectors. Unfortunately, some storage inefficiencies occur, because a 1 byte file now requires 1024 bytes of storage instead of 512 bytes if it is stored on a 3½ or 5¼ inch diskette that records data in clusters of 1 sector per cluster.

Next I tested the 40M byte hard disk in a COMPAQ 286 PC using DOS 3.2. Figure 7.14 shows that 26,648,576 bytes were originally available for storage, but when I created an 8 byte file called XXX.ZZZ, only 26,644,480 bytes were left. This is a difference of 4096 bytes. Here the reason why the COMPAQ PC uses a 4096 byte cluster of 4 sectors is the fact that, using DOS versions 3.0 and later, the hard disk can be set up for either a 2048 or 4096 byte cluster size. In this situation, the 40M byte disk of the COMPAQ PC was set up to record and retrieve data in units of 4 sectors per cluster, or a total of 4096 bytes of storage.

An IBM PS/2 Model 30 with a 30M byte hard disk was checked next as shown in Figure 7.15. Here the original bytes free were 13,082,624 before creation of a 7 byte file called XXX.ZZZ. After this file was created, only 13,080,576 bytes were free. This is a difference of 2048 bytes, another multiple of 512. Here the use of a 2048 byte cluster results from the method by which the hard disk was initialized.

The IBM PS/2 Model 30 also has a 3½ inch floppy, which can store 720K bytes. The test of the method of data storage on this media is shown in Figure 7.16. Note that 432,128 bytes were originally available, but after creating an 8 byte file called XXX.ZZZ, only

Figure 7.14

Determining the COMPAQ 286 Cluster Size

```
LOGON     BAS      1543   2-03-88   3:05p
USERS     DAT       103   2-03-88   3:10p
TAXABLE   BAT       335   2-03-88   2:22p
TAXNOT    BAT       326   2-03-88   2:23p
DIRSRCH   COM      4841   6-18-84  10:56a
SIZE      BAT        70   3-14-88   1:18p
DIR       DAT      1640   3-14-88   1:19p
SIZEOLD   BAT        93   3-10-88   2:21p
SIZE      BAK        70   3-14-88   1:18p
       42 File(s)  26648576 bytes free

C>copy con xxx.zzz
jfjfjf
^Z
        1 File(s) copied

C>dir *.zz

 Volume in drive C has no label
 Directory of  C:\BATUTIL

XXX       ZZZ         8   3-14-88   1:19p
        1 File(s)  26644480 bytes free

C>
```

Figure 7.15

Determining the IBM PS/2 Model 30 Cluster Size

```
SW       HLP    23690   2-03-86  12:43a
SW123VID EXE     3942  11-02-85  11:23a
STEVE    PIC     1049   9-08-87   9:55a
CPU      WK1     1769  10-14-87  10:24a
CPU      PIC     1065   9-08-87  10:09a
DASD     PIC    10929   9-14-87   9:40a
DASD     WK1     3168   9-14-87  10:40a
DASD1    PIC     7681   9-14-87  10:33a
DASD2    PIC     8383   9-14-87  10:41a
      60 File(s)   13082624 bytes free

C>copy con xxx.zzz
abcde
^Z
        1 File(s) copied

C>dir *.zzz

 Volume in drive C has no label
 Directory of  C:\123

XXX      ZZZ         7   3-15-88   9:40a
        1 File(s)  13080576 bytes free

C>
```

Figure 7.16

Determining the Cluster Size of an IBM PS/2 Model 30 720K Byte Diskette

```
HPPLUS    LAS     11006   7-31-86   5:17p
CANON     LAS      9739   8-28-86   2:41a
RTDEMO    EXE     36490  10-06-86   4:30p
SG_ASK    COM       592   9-14-86   3:28p
SG_ASKW   COM       724   9-14-86   3:31p
SG_DMODE  COM       460   9-11-86   9:21p
SG_MEM    COM       550   9-11-86   9:21p
SG_NOTES  COM       696   9-14-86   5:17p
DEMO      FIL     30479   1-01-80  12:20a
       16 File(s)    432128 bytes free

C:\PROCOMM>copy con a:xxx.zzz
abcdef
^Z
        1 File(s) copied

C:\PROCOMM>dir a:*.zzz

 Volume in drive A has no label
 Directory of  A:\

XXX       ZZZ         8   3-15-88   9:16a
        1 File(s)    431104 bytes free
```

Determining the Size of a Directory

431,104 bytes were free. This is a difference of 1024 bytes. This PS/2 Model 30 computer was running DOS 3.3.

To conclude this short series of tests, I next used an IBM PS/2 Model 50 personal computer operating under the control of DOS 3.3. This PC has a 1.4M byte floppy diskette drive, which I investigated to determine its cluster size. Figure 7.17 illustrates the test conducted to determine the minimum storage unit used on an IBM PS/2 Model 50 1.4M byte floppy diskette drive. Here there were 1,299,968 bytes free prior to creating a 5 byte file called XXX.ZZZ. Note that after the file was created, a new directory listing showed that only 1,299,456 bytes were free, a difference of 512 bytes.

Expanding the SIZE Command

A couple of problems are inherent with the SIZE command previously created. Because you may be using several different types of disk drives and different diskette formats, the amount of storage space available on a diskette will vary. You may even wish to copy a complete subdirectory onto a diskette that already has some files on it or that has some bad sectors marked during the formatting process. Either of these situations results in diskettes that do not always contain the same amount of available storage space.

Because diskettes may have 320K, 360K, 720K, 1.2M, or 1.4M bytes of storage, or some other number, we need a way to identify how many bytes are available on a particular diskette when we are ready to make the subdirectory copy. A simple answer to this is to use the DOS CHKDSK command. CHKDSK shows the total bytes of disk space, the number of bytes in user files, and the number of bytes available on disk for future storage. An example of the CHKDSK command is shown in Figure 7.18. Notice that in this example there are already two files on the disk, and they have taken 2048 bytes of the available 362,496 bytes of total disk space. We can use the in-

Figure 7.17

Determining the Cluster Size of an IBM PS/2 Model 50 1.4M Byte Diskette Drive

```
CHICAGO  DOC      1536  12-18-87   5:27p
DIAL-IN  DOC      2560  12-21-87   2:19p
SURPLUS  DOC      2048   2-03-88  11:27p
SAUTER   DOC      2560   2-03-88   2:27p
SURVEY   DOC      2048   2-11-88  11:42a
PIPSDIAL DOC      2560   2-11-88  11:42a
ASYNCH   DOC     15360   2-23-88   2:28p
PROTOCOL DOC     51712   2-29-88   2:52p
DONNA    DOC      4608   3-11-88   3:23p
       23 File(s)   1299968 bytes free

C>copy con a:xxx.zzz
abcde
^Z
        1 File(s) copied

C>dir a:*.zzz

 Volume in drive A has no label
 Directory of  A:\

XXX      ZZZ         7   3-15-88   9:41a
        1 File(s)   1299456 bytes free
```

Figure 7.18

Using the CHKDSK Command

```
C>chkdsk b:

   362496 bytes total disk space
     2048 bytes in 2 user files
   360448 bytes available on disk

   655360 bytes total memory
   426080 bytes free

C>
C>
```

formation from the CHKDSK command in our BASIC program SIZEPROG.BAS as the maximum amount of disk space on a diskette.

The first task is to modify the SIZE.BAT command to include the CHKDSK command and send the resulting output to a file that can be read by the SIZEPROG.BAS BASIC program. Program Listing 7.25 shows the modified SIZE.BAT program.

Program Listing 7.25

Modified SIZE.BAT File

```
1:*echo off
2: chkdsk %1 ¦ FIND "bytes available on disk" > chkdsk.dat
3: dir %2 ¦ FIND ":" ¦ FIND /V "<DIR>" ¦ FIND /V "Directory" > dir.dat
4: bsic sizeprog.bas
```

Notice that line 2 now contains the CHKDSK command, and CHKDSK uses the first parameter passed when the SIZE command is issued. The SIZE command must now use the format:

```
        SIZE  d: d: \subdirectory
where:   d:                  .......the drive to be copied to.
         d:\subdirectory  .......the subdirectory to be
                                 sized for fitting on the
                                 diskette in the %1 drive.
```

Line 2 of Program Listing 7.25 shows that the output from the CHKDSK command is passed to a FIND command that selects only the line from the command that contains the bytes available on disk and writes this line to a file called CHKDSK.DAT. Line 3 has been changed only in the fact that the DIR command now operates on the second parameter passed to the SIZE command. If a second parameter is not passed, the default is the current directory. The top of Program Listing 7.25 shows an example of the use of the SIZE command, where the disk to be written to is in the B: drive, and the subdirectory to be sized is the C: \DOS subdirectory. Notice that the result shows that the space required is 549,888 bytes, while the available space is only 360,448. Therefore, the subdirectory will not fit on the diskette in drive B. The lower portion of Figure 7.19 shows another example of the use of the SIZE command with just the destination diskette drive B specified. In this example the current directory is sized and the space required is 58,368 bytes, which will fit on the diskette.

Now let us examine the program SIZEPROG.BAS. Several changes were made to this program to handle the new data coming from the CHKDSK command. Program Listing 7.26 shows a listing of this program with these changes. Line 20 now opens the CHKDSK.DAT file and line 40 reads the first record into the string variable L$. Line 50 takes the number in the first 9 character positions of that string and converts it from a string to a numeric value, then keeps it in the variable D1#. Line 60 next closes the CHKDSK.DAT file as the only information in it that we needed was how much space was available

Figure 7.19 Using the Modified SIZE.BAT Program

```
C>size b:c:\DOS


not OK..
Total space required      =  549888
Available space on disk  =   360448


C>size b:

Files will fit on DISKETTE
Total space required      =  58368
Available space on disk  =   360448


C>
```

on the diskette. The remainder of the program is essentially unchanged, except that line 170 now tests to see if the total space required will fit in the space on the disk as recorded in the CHKDSK.DAT file and stored previously in the D1# variable. Line 190 also shows the amount of disk space as computed in line 50, which used the D1# variable to extract the number of bytes of available space from the CHKDSK.DAT file.

Program Listing 7.26 Modified SIZEPROG.BAS Program

```
10 OPEN "DIR.DAT" FOR INPUT AS #1
20 OPEN "CHKDSK.DAT" FOR INPUT AS #2
```

```
30 L$=""
40 LINE INPUT #2,L$
50 LET D1#=VAL(LEFT$(L$,9))
60 CLOSE #2
70 SZ1=1024
80 LET SUM#=0
90 LINE INPUT #1,L$
100 IF EOF(1) THEN GOTO 160
110 LET N#=VAL(MID$(L$,13,9))
120 LET X#=INT(N#/SZ1)+.9999)
130 LET SUM#=SUM#+X#
140 LET TOT#=SUM#*SZ1
150 GOTO 90
160 CLOSE #1
170 IF TOT#<D1# THEN PRINT "Files will fit on DISKETTE" ELSE PRINT "not OK.."
180 PRINT "Total space required     = ";TOT#
190 PRINT "Available space on disk  =  ";D1#
200 SYSTEM
210 END
```

An additional change worth mentioning is the sector size chosen in the previous SIZEPROG.BAS program. Notice that the 1024 sector size is not always valid, because in our earlier discussions we showed that many diskettes use a 512 byte sector size, and some of the hard disks use sector sizes up to 4096 bytes. Because most of our copying will be to a floppy disk, we can generally ignore the larger sector sizes, but we must have a good idea of what sector sizes are used by diskettes on the current system. To make the program more useful, we can add the computations to show results if the sector sizes are 512 or 1024.

Program Listing 7.27 contains the listing of the SIZEPROG.BAS BASIC program after modification to accommodate 512 and 1024 byte cluster sizes. Lines 70 and 80 designate the different sizes used and place them in variables SZ0 and SZ1, respectively. Lines 90 and 100 initialize a SUM0# and SUM1# as different summing variables for each of the space computations. Lines 140 through 190 perform the computations, one for each of the cluster sizes used. Lines 230 through 260 print the results for 512 byte computations, and lines 280 through 310 print the results for the 1024 byte sizes.

Program Listing 7.27

Modified Program to Consider Different Cluster Sizes

```
10 OPEN "DIR.DAT" FOR INPUT AS #1
20 OPEN "CHKDSK.DAT" FOR INPUT AS #2
30 L$=""
40 LINE INPUT #2,L$
50 LET D1#=VAL(LEFT$(L$,9))
60 CLOSE #2
70 LET SZ0=512
80 LET SZ1=1024
90 LET SUM0#=0
100 LET SUM1#=0
110 LINE INPUT #1,L$
120 IF EOF(1) THEN GOTO 210
130 LET N#=VAL(MID$(L$,13,9))
140 LET X0#=INT((N#/SZ0)+.9999)
150 LET X1#=INT((N#/SZ1)+.9999)
160 LET SUM0#=SUM0#+X0#
170 LET SUM1#=SUM1#+X1#
180 LET TOT0#=SUM0#*SZ0
190 LET TOT1#=SUM1#*SZ1
200 GOTO 110
210 CLOSE #1
220 PRINT
230 PRINT "For 512 byte cluster ..."
240 IF TOT0#<D1# THEN PRINT "Files will fit on DISKETTE" ELSE PRINT "not OK..."
250 PRINT "Total space required      = ";TOT0#
260 PRINT "Available space on disk  =  ";D1#
270 PRINT
280 PRINT "For 1024 byte cluster ..."
290 IF TOT1#<D1# THEN PRINT "Files will fit on DISKETTE" ELSE PRINT "not OK.."
300 PRINT "Total space required      = ";TOT1#
310 PRINT "Available space on disk  =  ";D1#
320 SYSTEM
330 END
```

Figure 7.20 shows an example of the results of running the SIZE command after making the previously discussed changes to the BASIC

Figure 7.20 **Determining the Space Requirements for Different Types of Media**

```
C>size b: \DOS



For 512 byte cluster ...
not OK...
Total space required          =  533504
Available space on disk       =   360448
For 1024 byte cluster ...
not OK..
Total space required          =  549888
Available space on disk       =   360448




C>
```

program. Notice that the results show space requirements for either 512 byte or 1024 byte cluster sizes.

SUPPRESSING MESSAGES

A problem often associated with batch files is the messages that result from the execution of certain DOS commands in the files. In this section we will examine the use of the NUL device address to send messages to the proverbial "bit bucket," a location where the NUL device can eliminate the possibility of confusing novice users with an unexpected screen display.

As many of our batch files have illustrated, you can use the ECHO OFF command to suppress the display of the contents of a batch file as each line executes. Unfortunately, ECHO OFF only suppresses the display of the contents of each line in the batch file, but not the DOS response to specific commands that may be included in the file. As an example of the preceding, consider the execution of the following two-line batch file:

```
ECHO OFF
COPY A:JAN.DAT B:
```

Although the ECHO OFF command suppresses the display of the line containing the COPY command, when that line is executed the resulting message `1 File(s) copied` will be displayed. For novice computer users, this message—as well as the display of other messages resulting from the execution of other DOS commands—could be confusing.

The key to suppressing messages is the DOS device address NUL. NUL is a nonexistent device address that, according to the DOS manual, can be used for testing. This means that you can direct the output of a command to NUL to test the execution of the command in a file without requiring the actual output from the command to occur.

One practical use for the NUL device address is in the development of a batch file that requires the printing of one or more files. Here the use of the command TYPE FILE.DAT>NUL would enable you to insert this command line into your batch file without having to generate a listing of the file each time you execute the program you are developing. Then, once you were satisfied that the program operates correctly, you could replace the device address NUL with the correct printer address, LPT1, LPT2, or LPT3.

A second practical use of the device address NUL is to suppress screen output. This is accomplished by using the I/O redirection feature of DOS in conjunction with the NUL device address for each command in your batch file that generates a message you wish to suppress. Returning to the COPY command example covered earlier in this section, let us modify that command:

```
COPY A:JAN.DAT B: > NUL
```

Here, the I/O redirection to device address NUL suppresses the message `1 File(s) copied.`

Although the preceding example only suppressed a one-line message, there are many situations in which the execution of a command can result in the display of several lines of text. Examples of commands in this category include the execution of commercial utility programs that display copyright notices, such as the Safepark program from Prime Solutions Inc. Figure 7.21 illustrates the resulting display when the command `SAFEPARK` is executed. If your batch file contains several commands that result in the display of messages you can eliminate their display by redirecting I/O to the device address NUL for each line. Thus, changing the command line `SAFEPARK` to `SAFEPARK`

Figure 7.21

Loading Safepark

```
C>SAFEPARK
Safepark Rev 02.10
(c) 1987 PRIME SOLUTIONS INC.
Safepark is installed and active
```

`> NUL` would suppress the three lines that the command normally generates.

As I have indicated, using the NUL device address provides you with a mechanism to suppress messages. By incorporating this technique into other techniques covered in this book, you have the ability to tailor your display to your intended audience.

Index

A

AND operator, 213-218
ANSI control codes, 126, 129, 201
ANSI.SYS file, 126-128, 129-131, 136, 201, 203, 204
ASCII code, 171, 194-195
audio generation, 194-196
AUTOEXEC.BAT file, 60-61, 109-110, 117-118, 230, 232, 247-248

B

background color, 128-135
BASIC, 106-117, 157-171, 212-213, 218-229, 230-235
batch commands, 49-61
batch file creation, 46-48
batch file operations, 45-61
BEL character, 195-196
BIO.COM file, 9
buffer, 148

C

CALC command, 219-229
calculator creation, 218-229
CALL command, 49, 60
CD command (*see* CHDIR command)
character attributes, 130
CHDIR command, 36-38, 105
CHKDSK command, 88-94
cluster size, 265-277
color display control, 128-135
COMMAND.COM file, 9, 202-203
command creation, 96-101
compound interest, 222-226
compressed print, 152, 173-177, 200-201
computer usage log, 229-239
condensed print, 152, 173-176, 200-201
CONFIG.SYS file, 129, 201, 204, 245-246
configuration file, 127, 129, 204, 245-246
COPY command, 46-48, 91, 98-99, 148, 151, 240-242

Page 81

cursor positioning, 136-138

D

DEL command, 98-99
device driver, 127, 129, 203
device fault, 181
device names, 10, 70-71
directory
  label generation, 198-202
  location, 93-96
  navigation, 121-123
  security, 74-75
  size, 256-257
  sorting, 76-83
directory label generation, 198-202

DISKCOPY command, 240-242
diskette capacity, 257-264
disk fragmentation, 240-246
DISPLAY command, 96-98
DOS
  command control keys, 20-22
  components, 8-10
  database, 188-193
  evolution, 6-7
  printer control, 171-179
  reserved names, 12-13
  versions, 7-8
dot command, 49, 51
doublestrike print mode, 152, 173-174, 176-178
doublewide printing, 173
draft elite print style, 156, 168
draft pica print style, 155, 168
drive identifier, 14, 16

E

ECHO command, 52, 69, 82, 93-95, 148-149
EDLIN, 47, 71-72, 82
emphasized print mode, 152, 165, 167-168, 173-178
Epson printer control codes, 152-156, 173
ERASE command, 98
erase help file, 186-187
ERRORLEVEL, 56, 249-252
Escape character, 127-128, 171-172, 206
evolution of DOS, 6-7
exit code processing, 249-252
EXIT command, 110-111
expanded-print mode, 152
extended code, 205
extensions, 11-15, 79-80

F

file allocation table, 240
file locator, 88-96
filenames, 11-13
filenaming conventions, 13-14
file references, 38-40
file size, 80-81
filters, 26-29
FIND command, 28-29, 91, 93-94, 122-123, 191-193, 199-200, 214-216, 220, 237-238
FOR command, 53-54
foreground color, 129-133
FORMAT.EXE file, 68-70, 242-244
format protection, 67-70
formula creation, 223-229
function keys, 204-207
future value calculation, 224-226

G

global filename symbols, 19-20
GOTO command, 54-55
graphics generation, 196-198

H

help files, 184-188
hexadecimal, 221
hidden files, 9
hierarchical directory structure, 16-19, 31-43
  construction, 33
  file references, 38-40
  purpose, 32

I

IBMBIO.COM file, 9
IBMDOS.COM file, 9
IF command, 55-58, 97-99, 185, 210, 242
illegal filenames, 202-203
illegal subdirectory names, 202-203
input redirection, 23-24
I/O redirection, 22-29
italic printing, 166-168

K

keyboard macro creation, 203-210

L

label generator, 147-151
letter quality print style, 154

M

mailing list creation, 147-151
mapping keys, 205-207
MD command (*see* MKDIR command)
menu creation, 99-123
menu selection logging, 235-239
message suppression, 278-280
MKDIR command, 34-36
MORE command, 27, 123
MOVE command, 98-101
MSDOS.COM, 9
multitasking, 147

N

near letter quality printing, 152

O

on-line assistance, 187-188
OR operator, 213-219
output redirection, 24-25

P

parallel printer port, 179-181
PATH command, 40-41, 109-110
paths, 16-19
PAUSE command, 52-53, 85-88

PEEK command, 141-142
pipes, 26-29
PLAY command, 163
PRINT command, 143-147
printer control, 139-181
printer mode control, 152-178
printer port swapping, 140-142
printer port testing, 179-180
printer spooling, 142-147
PROMPT command, 103-104, 120, 129-138, 206-209
prompts, 10
proportional print mode, 153, 165, 167-168

R

RAM disk, 245-248
RD command (*see* RMDIR command)
RECOVER command, 245
REM command, 50-51
REM statement, 107
RENAME command, 202
replaceable parameters, 48-49
reserved extensions, 14
reserved names, 12-13
RMDIR command, 42-43
root directory, 16

S

screen control, 125-138
SEARCH command, 213-218
sectors, 239-241
security, 74-76
  file, 75-76
  subdirectory, 75-76
session duration, 238-239
SHELL command, 108-109, 113
SHIFT command, 58-59
signon program, 230-235
single drive copying, 71-74
SORT command, 26-29, 190-193
sorting menu, 83-88
spooling, 142-145
standard I/O, 22-23
string variable, 116
SYSTEM command, 112-113, 220

T

telephone directory, 189-193
text erasure, 137-138
text searching, 213-218
TIME command, 210-211, 256
time processing, 210-213, 255-256
TREE command, 41-42, 122-123

U

utility menu, 120-121
utility routines, 183-280

V

VDISK program, 246
versions of DOS, 7-8

W

wide directory, 83-84, 87-88
wildcards, 19

COMPUTERS/DOS $22.95

Why invest thousands of dollars in software when you can unlock the built-in power of DOS?

DOS PRODUCTIVITY TIPS AND TRICKS

Now, you can have instant access to DOS programs you might have thought were a fortune in software away. *DOS Productivity: Tips & Tricks* gives you all the tools you need to develop over 20 DOS routines on your own—all virtually identical to commercial software in function and all without the help of stand-alone application programs. From overviews of DOS and batch files characteristics to the total operation of file and subdirectories, here is one DOS guide that will pay for itself hundreds of times over. Extensive coverage includes:

- Special techniques like creating your own DOS commands, time-dependent program execution, usage logging, master menu creation, and instant database applications.
- Important operational techniques such as how to avoid accidentally reformatting a disk, and how to prevent other users from accessing your files and directories.
- Tips on screen control including setting colors, display switching, and screen dumps.
- Printing methods for changing printer modes, port swapping, and creating labels and mailing.

Packed with hundreds of productivity illustrations you can adapt to your own PC work, *DOS Productivity: Tips & Tricks* is the most cost-efficient way to maximize the tremendous power of DOS.

GILBERT HELD is director of 4-Degree Consulting. He is the author of *The PC Upgrader's Manual, The IBM PC-XT User's Manual* and *Data Communications Networking Devices*. Mr. Held holds an MBA in operations research and an MSTM in computer science from American University. He earned his MSEE at New York University and is the only person to twice receive the coveted Karp award for excellence in technical writing.

JOHN WILEY & SONS
Business/Law/General Books Division
605 Third Avenue, New York, N.Y. 10158-0012
New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0-471-60895-5
52295
9 780471 608950

Cover Design: *Laurie Angel-Sadis*